The SiLK Reference Guide
(SiLK-3.22.0)

CERT Software Automation Product Development
©2002–2023 Carnegie Mellon University
License available in Appendix A

The canonical location for this handbook is
https://tools.netsa.cert.org/silk/silk-reference-guide.pdf

September 14, 2023

Contents

Introduction **7**

1 SiLK Analysis Tools and Utilities **9**

mapsid . . . 10
num2dot . . . 14
rwaddrcount . . . 17
rwaggbag . . . 27
rwaggbagbuild . . . 34
rwaggbagcat . . . 43
rwaggbagtool . . . 53
rwappend . . . 63
rwbag . . . 66
rwbagbuild . . . 78
rwbagcat . . . 92
rwbagtool . . . 105
rwcat . . . 118
rwcombine . . . 122
rwcompare . . . 130
rwcount . . . 133
rwcut . . . 141
rwdedupe . . . 156
rwfglob . . . 163
rwfileinfo . . . 170
rwfilter . . . 177
rwgeoip2ccmap . . . 205
rwgroup . . . 216
rwidsquery . . . 225
rwip2cc . . . 229
rwipaexport . . . 234
rwipaimport . . . 237
rwipfix2silk . . . 239
rwmatch . . . 244
rwnetmask . . . 255
rwp2yaf2silk . . . 260
rwpcut . . . 262
rwpdedupe . . . 264
rwpdu2silk . . . 266
rwpmapbuild . . . 270
rwpmapcat . . . 281
rwpmaplookup . . . 290

rwpmatch . . . 302
rwptoflow . . . 304
rwrandomizeip . . . 311
rwrecgenerator . . . 314
rwresolve . . . 324
rwscan . . . 327
rwscanquery . . . 336
rwset . . . 345
rwsetbuild . . . 351
rwsetcat . . . 356
rwsetmember . . . 370
rwsettool . . . 372
rwsilk2ipfix . . . 383
rwsiteinfo . . . 393
rwsort . . . 406
rwsplit . . . 416
rwstats . . . 421
rwswapbytes . . . 444
rwtotal . . . 447
rwtuc . . . 454
rwuniq . . . 464
silk_config . . . 486

3 SiLK Libraries and Plug-Ins **489**
addrtype . . . 490
app-mismatch . . . 493
ccfilter . . . 496
conficker-c . . . 499
cutmatch . . . 503
flowkey . . . 506
flowrate . . . 511
int-ext-fields . . . 515
ipafilter . . . 520
packlogic-generic.so . . . 522
packlogic-twoway.so . . . 526
pmapfilter . . . 531
PySiLK . . . 538
silk-plugin . . . 575
silkpython . . . 596

5 SiLK File Formats **627**
sensor.conf . . . 628
silk.conf . . . 650

7 SiLK Miscellaneous Information **657**
SiLK . . . 658

8 SiLK Administrator's Tools **669**
flowcap . . . 670
rwflowappend . . . 676
rwflowpack . . . 683
rwguess . . . 698

rwpackchecker 702
rwpollexec 708
rwreceiver 713
rwsender 721

A License **729**

Introduction

The *SiLK Reference Guide* contains the manual page for each analysis tool, utility, plug-in, file format, and collection facility in the SiLK Collection and Analysis Suite.

This document is meant for reference only. The *SiLK Analysis Handbook* provides both a tutorial for learning about the tools and examples of how they can be used in analyzing flow data. See the *SiLK Installation Handbook* for instructions on installing SiLK at your site.

This reference guide is broken into sections like the traditional UNIX manual: end-user analysis tools and utilities are described in Section 1; the libraries and plug-ins that augment the behavior of some tools are presented in Section 3; Section 5 contains information about file formats; miscellaneous information is in Section 7; and commands for the installer and administrator of SiLK appear in Section 8.

1

SiLK Analysis Tools and Utilities

This section provides the manual page for each analysis tool and utility that the users of SiLK may employ in their day-to-day work.

mapsid

Map between sensor names and sensor numbers

SYNOPSIS

```
mapsid [--print-classes] [--print-descriptions]
      [--site-config-file=FILENAME]
      [{ <sensor-name> | <sensor-number> } ...]

mapsid --help

mapsid --version
```

DESCRIPTION

As of SiLK 3.0, **mapsid** is deprecated, and it will be removed in the SiLK 4.0 release. Use **rwsiteinfo(1)** instead---the EXAMPLES section shows how to use **rwsiteinfo** to get output similar to that produced by **mapsid**.

mapsid is a utility that maps sensor names to sensor numbers or vice versa depending on the input arguments. Sensors are defined in the **silk.conf(5)** file.

When no sensor arguments are given to **mapsid**, the mapping of all sensor numbers to names is printed. When a numeric argument is given, the number to name mapping is printed for the specified argument. When a name is given, its numeric id is printed. For convenience when typing in sensor names, case is ignored.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--print-classes

For each sensor, print the classes for which the sensor collects data. The classes are enclosed in square brackets, `[]`.

--print-descriptions

For each sensor, print the description of the sensor as defined in the *silk.conf* file (if any).

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **mapsid** searches for the site configuration file in the locations specified in the FILES section.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

The following examples demonstrate the use of **mapsid**. In addition, each example shows how to get similar output using **rwsiteinfo(1)**.

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

Name to number mapping

```
$ mapsid beta
BETA ->     1
```

```
$ rwsiteinfo --fields=sensor,id-sensor --sensors=BETA
Sensor|Sensor-ID|
  BETA|        1|
```

Unlike **mapsid**, matching of the sensor name is case-sensitive in **rwsiteinfo**.

Number to name mapping

```
$ mapsid 3
    3 -> DELTA
```

```
$ rwsiteinfo --fields=id-sensor,sensor --sensors=3 --delimited=,
Sensor-ID,Sensor
3,DELTA
```

Print all mappings

```
$ mapsid
    0 -> ALPHA
    1 -> BETA
    2 -> GAMMA
    3 -> DELTA
    4 -> EPSLN
    5 -> ZETA
     ....
```

```
$ rwsiteinfo --fields=id-sensor,sensor --no-titles
  0| ALPHA|
  1|  BETA|
  2| GAMMA|
  3| DELTA|
  4| EPSLN|
  5|  ZETA|
  ...
```

Print the class

```
$ mapsid --print-classes 3 ZETA
    3 -> DELTA  [all]
ZETA  ->     5  [all]

$ rwsiteinfo --fields=id-sensor,sensor,class:list --sensors=4,ZETA
Sensor-ID|Sensor|Class:list|
        3| DELTA|       all|
        5|  ZETA|       all|
```

Print the class and description

```
$ mapsid --print-classes --print-description 0 1
    0 -> ALPHA  [all]  "Primary gateway"
    1 -> BETA   [all]  "Secondary gateway"
```

rwsiteinfo supports using an integer range when specifying sensors.

```
$ rwsiteinfo --fields=id-sensor,sensor,class:list,describe-sensor \
      --sensors=0-1
Sensor-ID|Sensor|Class:list|Sensor-Description|
        0| ALPHA|       all|   Primary gateway|
        1|  BETA|       all| Secondary gateway|
```

ENVIRONMENT

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **mapsid** may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **mapsid** may use this environment variable. See the FILES section for details.

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

SEE ALSO

rwsiteinfo(1), **silk.conf(5)**, **silk(7)**

NOTES

As of SiLK 3.0, **mapsid** is deprecated; use **rwsiteinfo(1)** instead.

num2dot

Convert an integer IP to dotted-decimal notation

SYNOPSIS

```
num2dot [--ip-fields=FIELDS] [--delimiter=C]

num2dot --help

num2dot --version
```

DESCRIPTION

num2dot is a filter to speedup sorting of IP numbers and yet result in both a `natural` order (i.e., 29.23.1.1 will appear before 192.168.1.1) and readable output (i.e., dotted decimal rather than an integer representation of the IP number).

It is designed specifically to deal with the output of **rwcut(1)**. Its job is to read stdin and convert specified fields (default field 1) separated by a delimiter (default '|') from an integer number into a dotted decimal IP address. Up to three IP fields can be specified via the **--ip-fields**=*FIELDS* option. The **--delimiter** option can be used to specify an alternate delimiter.

num2dot does not support IPv6 addresses. The EXAMPLES section below includes an example PySiLK script to handle IPv6.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--ip-fields=*FIELDS*

Column number of the input that should be considered IP numbers. Column numbers start from 1. If not specified, the default is 1.

--delimiter=*C*

The character that separates the columns of the input. Default is '|'.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following example, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

Suppose in addition to the default fields of 1-12 produced by **rwcut(1)**, you want to prefix each row with an integer form of the destination IP and the start time to make processing by another tool (e.g., a spreadsheet) easier. However, within the default **rwcut** output fields of 1-12, you want to see dotted-decimal IP addresses. You could use the following command:

```
$ rwfilter ... --pass=stdout                                  \
  | rwcut --fields=dip,stime,1-12 --ip-format=decimal         \
        --timestamp-format=epoch                              \
  | num2dot --ip-field=3,4
```

In the **rwcut** invocation, you prepend the fields of interest (`dip` and `stime` before the standard fields. The first six columns produced by **rwcut** will be dIP, sTime, sIP, dIP, sPort, dPort. The **--ip-format** switch causes the first, third, and fourth columns to be printed as integers, but you only want the first column to have an integer representation. The pipe through **num2dot** will convert the third and fourth columns to dotted-decimal IP numbers.

num2dot does not support converting integers to IPv6 addresses. The following PySiLK script (see **pysilk(3)**) could be used as a starting-point to create a version of **num2dot** that supports IPv6 addresses:

```
#! /usr/bin/env python
from __future__ import print_function
import sys
import silk
# The IPv6 fields to process; the ID of the first field is 0
ip_fields = (0, 1)
# The delimiter between fields
delim = '|'
# The width of the IPv6 fields
width = 39
# The file to process; this script processes standard input
f = sys.stdin
try:
    for line in f:
        fields = line.rstrip(f.newlines).split(delim)
        for i in ip_fields:
            fields[i] = "%*s" % (width, silk.IPv6Addr(int(fields[i])))
        print(delim.join(fields))
finally:
    f.close()
```

SEE ALSO

rwcut(1), **pysilk(3)**, **silk(7)**

BUGS

num2dot has no support for IPv6 addresses.

rwaddrcount

Count activity by IPv4 address

SYNOPSIS

```
rwaddrcount {--print-recs | --print-ips | --print-stat}
      [--use-dest] [--min-bytes=BYTEMIN] [--max-bytes=BYTEMAX]
      [--min-records=RECMIN] [--max-records=RECMAX]
      [--min-packets=PACKMIN] [--max-packets=PACKMAX]
      [--set-file=PATHNAME] [--sort-ips] [--timestamp-format=FORMAT]
      [--ip-format=FORMAT] [--integer-ips] [--zero-pad-ips]
      [--no-titles] [--no-columns] [--column-separator=CHAR]
      [--no-final-delimiter] [{--delimited | --delimited=CHAR}]
      [--print-filenames] [--copy-input=PATH] [--output-path=PATH]
      [--pager=PAGER_PROG] [--site-config-file=FILENAME]
      [{--legacy-timestamps | --legacy-timestamps=NUM}]
      {[--xargs] | [--xargs=FILENAME] | [FILE [FILE ...]]}

rwaddrcount --help

rwaddrcount --version
```

DESCRIPTION

rwaddrcount reads SiLK Flow records, sums the byte-, packet-, and record-counts on those records by individual source or destination IP address and maintains the time window during which that IP address was active. At the end of the count operation, the results per IP address are displayed when the **--print-recs** switch is given. **rwaddrcount** includes facilities for displaying only those IP address whose byte-, packet- or flow-counts are between specified minima and maxima.

rwaddrcount does not support IPv6 addresses. To generate output for IPv6 records, use the **rwuniq(1)** tool:

```
rwuniq --fields=sip --values=bytes,packets,records,stime,etime
```

rwaddrcount reads SiLK Flow records from the files named on the command line or from the standard input when no file names are specified and **--xargs** is not present. To read the standard input in addition to the named files, use `-` or `stdin` as a file name. If an input file name ends in `.gz`, the file is uncompressed as it is read. When the **--xargs** switch is provided, **rwaddrcount** reads the names of the files to process from the named text file or from the standard input if no file name argument is provided to the switch. The input to **--xargs** must contain one file name per line.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

For the application to operate, one of the three **--print** options must be chosen.

--print-recs

Print one row for each bin that meets the minima/maxima criteria. Each bin contains the IP address, number of bytes, number of packets, number of flow records, earliest start time, and latest end time.

--print-ips

Print a single column containing the IP addresses for each bin that meets the minima/maxima criteria.

--print-stat

Print a one or two line summary (plus a title line) that summarizes the bins. The first line is a summary across **all** bins, and it contains the number of unique IP addresses and the sums of the bytes, packets, and flow records. The second line is printed only when one or more minima or maxima are specified. This second line contains the same columns as first, and its values are the sums across those bins that meet the criteria.

--use-dest

Count by destination IP address in the filter record rather than source IP.

--min-bytes=*BYTEMIN*

Filtering criterion; for the final output (stats or printing), only include count records where the total number of bytes exceeds BYTEMIN

--min-packets=*PACKMIN*

Filtering criterion; for the final output (stats or printing), only include count records where the total number of packets exceeds PACKMIN

--min-records=*RECMIN*

Filtering criterion; for the final output (stats or printing), only include count records where the total number of filter records contributing to that count record exceeds RECMIN.

--max-bytes=*BYTEMAX*

Filtering criterion; for the final output (stats or printing), only include count records where the total number of bytes is less than BYTEMAX.

--max-packets=*PACKMAX*

Filtering criterion; for the final output (stats or printing), only include count records where the total number of packets is less than PACKMAX.

--max-records=*RECMAX*

Filtering criterion; for the final output (stats or printing), only include count records which at most RECMAX filter records contributed to.

--set-file=*PATHNAME*

Write the IPs into the **rwset(1)**-style binary IP-set file named PATHNAME. Use **rwsetcat(1)** to see the contents of this file.

--timestamp-format=*FORMAT*

Specify the format and/or timezone to use when printing timestamps. When this switch is not specified, the SILK_TIMESTAMP_FORMAT environment variable is checked for a default format and/or timezone. If it is empty or contains invalid values, timestamps are printed in the default format, and the timezone is UTC unless SiLK was compiled with local timezone support. *FORMAT* is a comma-separated list of a format and/or a timezone. The format is one of:

default
Print the timestamps as *YYYY/MM/DD*T*hh*:*mm*:*ss*

iso
Print the timestamps as *YYYY-MM-DD hh*:*mm*:*ss*

m/d/y
Print the timestamps as *MM/DD/YYYY hh*:*mm*:*ss*

epoch
Print the timestamps as the number of seconds since 00:00:00 UTC on 1970-01-01.

When a timezone is specified, it is used regardless of the default timezone support compiled into SiLK. The timezone is one of:

utc
Use Coordinated Universal Time to print timestamps.

local
Use the TZ environment variable or the local timezone.

--ip-format=*FORMAT*

For the **--print-recs** and **--print-ips** output formats, specify how IP addresses are printed, where *FORMAT* is a comma-separated list of the arguments described below. When this switch is not specified, the SILK_IP_FORMAT environment variable is checked for a value and that format is used if it is valid. The default *FORMAT* is `canonical`. *Since SiLK 3.7.0.*

canonical
Print IP addresses in the canonical format: dot-separated decimal for IPv4 (`192.0.2.1`).

no-mixed
Print IP addresses in the canonical format (`192.0.2.1`). Prevent use of the mixed IPv4-IPv6 representation when `map-v4` is also included in *FORMAT*. For example, use `::ffff:c000:201` instead of `::ffff:192.0.2.1`. *Since SiLK 3.17.0.*

decimal
Print IP addresses as integers in decimal format. For example, print `192.0.2.1` and `::ffff:192.0.2.1` as `3221225985` and `281473902969345`, respectively.

hexadecimal
Print IP addresses as integers in hexadecimal format. For example, print `192.0.2.1` and `::ffff:192.0.2.1` as `c00000201` and `ffffc00000201`, respectively.

zero-padded
Make all IP address strings contain the same number of characters by padding numbers with leading zeros. For example, print `192.0.2.1` as `192.000.002.001`. For IPv6 addresses, this setting implies `no-mixed`, so that `::ffff:192.0.2.1` is printed as `0000:0000:0000:0000:0000:ffff:c000:0201`. As of SiLK 3.17.0, may be combined with any of the above, including `decimal` and `hexadecimal`.

The following arguments modify certain IP addresses prior to printing. These arguments may be combined with the above formats.

map-v4
Change addresses to IPv4-mapped IPv6 addresses (addresses in the ::ffff:0:0/96 netblock) prior to formatting. *Since SiLK 3.17.0.*

unmap-v6
Do nothing (**rwaddrcount** does not support IPv6 addresses as the key). *Since SiLK 3.17.0.*

The following argument is also available:

force-ipv6
Set *FORMAT* to `map-v4,no-mixed`.

--integer-ips

Print IP addresses as integers. This switch is equivalent to **--ip-format=decimal**, it is deprecated as of SiLK 3.7.0, and it will be removed in the SiLK 4.0 release.

--zero-pad-ips

Print IP addresses as fully-expanded, zero-padded values in the canonical format. This switch is equivalent to **--ip-format=zero-padded**, it is deprecated as of SiLK 3.7.0, and it will be removed in the SiLK 4.0 release

--sort-ips

For the **--print-recs** and **--print-ips** output formats, the results are presented sorted by IP address.

--no-titles

Turn off column titles. By default, titles are printed.

--no-columns

Disable fixed-width columnar output.

--column-separator=*C*

Use specified character between columns and after the final column. When this switch is not specified, the default of '|' is used.

--no-final-delimiter

Do not print the column separator after the final column. Normally a delimiter is printed.

--delimited

--delimited=*C*

Run as if **--no-columns --no-final-delimiter --column-sep**=*C* had been specified. That is, disable fixed-width columnar output; if character *C* is provided, it is used as the delimiter between columns instead of the default '|'.

--print-filenames

Print to the standard error the names of input files as they are opened.

--copy-input=*PATH*

Copy all binary SiLK Flow records read as input to the specified file or named pipe. *PATH* may be `stdout` or `-` to write flows to the standard output as long as the **--output-path** switch is specified to redirect **rwaddrcount**'s textual output to a different location.

--output-path=*PATH*

Write the textual output to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output (and bypass the paging program). If *PATH* names an existing file, **rwaddrcount** exits with an error unless the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. If this switch is not given, the output is either sent to the pager or written to the standard output.

--pager=*PAGER_PROG*

When output is to a terminal, invoke the program *PAGER_PROG* to view the output one screen full at a time. This switch overrides the SILK_PAGER environment variable, which in turn overrides the PAGER variable. If the **--output-path** switch is given or if value of the pager is determined to be the empty string, no paging is performed and all output is written to the terminal.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwaddrcount** searches for the site configuration file in the locations specified in the FILES section.

--legacy-timestamps

--legacy-timestamps=*NUM*

When *NUM* is not specified or is 1, this switch is equivalent to **--timestamp-format=m/d/y**. Otherwise, the switch has no effect. This switch is deprecated as of SiLK 3.0.0, and it will be removed in the SiLK 4.0 release.

--xargs

--xargs=*FILENAME*

Read the names of the input files from *FILENAME* or from the standard input if *FILENAME* is not provided. The input is expected to have one filename per line. **rwaddrcount** opens each named file in turn and reads records from it as if the filenames had been listed on the command line.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

Deprecated Switches

The following switches are deprecated. They will be removed in SiLK 4.0.

--byte-min=*BYTEMIN*

Deprecated alias for **--min-bytes**.

--packet-min=*PACKMIN*

Deprecated alias for **--min-packets**.

--rec-min=*RECMIN*

Deprecated alias for **--min-records**.

--byte-max=*BYTEMAX*

Deprecated alias for **--max-bytes**.

--packet-max=*PACKMAX*

Deprecated alias for **--max-packets**.

--rec-max=*RECMAX*

Deprecated alias for **--max-records**.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

To print a list of source IP addresses that appeared in exactly one TCP record during the first 12 hours of 2003-Sep-01, use:

```
$ rwfilter --start-date=2003/09/01:00 --end-date=2003/09/01:11     \
       --proto=6 --pass=stdout                                     \
  | rwaddrcount --max-records=1 --print-ips
```

In general, to print out record information, use **rwaddrcount** with **--print-recs**

```
$ rwfilter --start-date=2003/01/17:00 --end-date=2003/01/17:23     \
       --proto=6 --pass=stdout                                     \
  | rwaddrcount --print-rec --no-title | head -3

 10.10.10.1|  65792| 147|  21| 2003/01/17T00:19:01| 2003/01/17T02:00:13|
 10.10.10.2| 110744|  89|   7| 2003/01/17T01:21:42| 2003/01/17T01:39:21|
 10.10.10.3|    864|  18|   6| 2003/01/17T00:20:33| 2003/01/17T01:25:38|
```

Replacements for rwaddrcount

We note some overlapping features between **rwaddrcount** and **rwuniq(1)**. There is often more than one way to perform the same task in the SiLK tool set.

Here's a guide to replacing each of the outputs of **rwaddrcount**:

The **--print-recs** switch prints five pieces of information for each source or destination address:

```
$ rwaddrcount --print-recs data.rw
          sIP|Bytes|Packets|Records|         Start_Time|           End_Time|
   10.0.0.144| 1646|      4|      1|2007/05/09T18:01:41|2007/05/09T18:01:41|
10.14.203.121|   40|      1|      1|2007/05/09T18:31:54|2007/05/09T18:31:54|
10.14.203.122|   40|      1|      1|2007/05/09T18:32:43|2007/05/09T18:32:43|
   10.15.6.14|  539|      3|      3|2007/05/09T18:03:05|2007/05/09T18:08:07|
  12.0.101.22| 4365|     23|      2|2007/05/09T18:26:43|2007/05/09T18:43:46|
```

To do the same in **rwuniq**, specify either `sip` in **--fields** and the **--values** shown here:

```
$ rwuniq --fields=sip --values=bytes,packets,flows,stime,etime data.rw
          sIP|Bytes|Packets|Records|          min_sTime|          max_eTime|
   10.0.0.144| 1646|      4|      1|2007/05/09T18:01:41|2007/05/09T18:01:41|
10.14.203.121|   40|      1|      1|2007/05/09T18:31:54|2007/05/09T18:31:54|
10.14.203.122|   40|      1|      1|2007/05/09T18:32:43|2007/05/09T18:32:43|
   10.15.6.14|  539|      3|      3|2007/05/09T18:03:05|2007/05/09T18:08:07|
  12.0.101.22| 4365|     23|      2|2007/05/09T18:26:43|2007/05/09T18:43:46|
```

When **rwaddrcount** includes **--use-dest**, change the **--fields** switch of **rwuniq** to `dip`. Replace the **--sort-ips** switch of **rwaddrcount** with **--sort-output** in **rwuniq**.

The **--print-stat** switch in **rwaddrcount** prints a one-line summary of the data:

```
$ rwaddrcount --print-stat data.rw
          |  sIP_Uniq|         Bytes|    Packets|   Records|
    Total|     57727|     948620676|    2026581|    382578|
```

This is difficult to produce with **rwuniq**. If there is a field that you know is either empty or constant across all records (such as `nhip` or `in`), you can use that as the key field in **rwuniq**.

```
$ rwuniq --fields=nhIP --values=distinct:sip,bytes,packets,flows data.rw
           nhIP|sIP-Distinct|        Bytes|   Packets|   Records|
        0.0.0.0|       57727|    948620676|   2026581|    382578|
```

Note that `class` generally does not work since each type within a class produces its own row:

```
$ rwuniq --fields=class --values=distinct:sip,bytes,packets,flows data.rw
class|sIP-Distinct|        Bytes|   Packets|   Records|
  all|        8674|    260143344|    964621|    151447|
  all|       55540|    688477332|   1061960|   6184399|
```

One trick is to use `stime` as the key with a very large **--bin-time**:

```
$ rwuniq --fields=stime --bin-time=2147483647             \
       --values=distinct:sip,bytes,packets,flows data.rw
              sTime|sIP-Distinct|        Bytes|   Packets|   Records|
1970/01/01T00:00:00|       57727|    948620676|   2026581|    382578|
```

Finally, you can use separate invocations of **rwfilter(1)**, **rwset(1)**, and **rwsetcat(1)**:

```
$ rwfilter --print-volume --all=stdout data.rw  \
  | rwset --sip=stdout                          \
  | rwsetcat --count-ips
     |      Recs|   Packets|        Bytes|     Files|
Total|    382578|   2026581|    948620676|         1|
 Pass|    382578|   2026581|    948620676|          |
 Fail|         0|         0|            0|          |
57727
```

rwaddrcount's **--print-ips** switch prints the IP addresses as text:

```
$ rwaddrcount --print-ips data.rw
            sIP
     10.0.0.144
  10.14.203.121
  10.14.203.122
     10.15.6.14
    12.0.101.22
```

A combination of **rwset** and **rwsetcat** is the best way to handle this:

```
$ rwset --sip-file=stdout data.rw | rwsetcat --print-ips
10.0.0.144
10.14.203.121
10.14.203.122
10.15.6.14
12.0.101.22
```

Alternatively, use **rwuniq** and the UNIX tool **cut(1)** to only print the first column:

```
$ rwuniq --fields=sIP data.rw  \
  | cut -d '|' -f 1
            sIP
     10.0.0.144
  10.14.203.121
  10.14.203.122
     10.15.6.14
    12.0.101.22
```

rwaddrcount allows you to restrict the output to bins that have a certain minimum or maximum count of bytes, packets, or flows via **--min-bytes**, **--max-bytes**, **--min-packets**, **--max-packets**, **--min-records**, and **--max-records**:

```
$ rwaddrcount --print-recs --min-byte=1024 --max-byte=2048 \
       --max-records=1 data.rw
          sIP|Bytes|Packets|Records|         Start_Time|           End_Time|
   10.0.0.144| 1646|      4|      1|2007/05/09T18:01:41|2007/05/09T18:01:41|
10.14.203.121|   40|      1|      1|2007/05/09T18:31:54|2007/05/09T18:31:54|
10.14.203.122|   40|      1|      1|2007/05/09T18:32:43|2007/05/09T18:32:43|
```

rwuniq supports the same operations using the **--bytes**, **--packets**, and **--flows** switches, each of which allows you to define a desired minimum and maximum value.

```
$ rwuniq --fields=sip --values=bytes,packets,records,stime,etime \
       --bytes=1024-2048 --flows=1-1 data.rw
         sIP|Bytes|Packets|Records|          min_sTime|          max_eTime|
   10.0.0.144| 1646|      4|      1|2007/05/09T18:01:41|2007/05/09T18:01:41|
10.14.203.121|   40|      1|      1|2007/05/09T18:31:54|2007/05/09T18:31:54|
10.14.203.122|   40|      1|      1|2007/05/09T18:32:43|2007/05/09T18:32:43|
```

ENVIRONMENT

SILK_IP_FORMAT

This environment variable is used as the value for **--ip-format** when that switch is not provided. *Since SiLK 3.11.0.*

SILK_TIMESTAMP_FORMAT

This environment variable is used as the value for **--timestamp-format** when that switch is not provided. *Since SiLK 3.11.0.*

SILK_PAGER

When set to a non-empty string, **rwaddrcount** automatically invokes this program to display its output a screen at a time. If set to an empty string, **rwaddrcount** does not automatically page its output.

PAGER

When set and SILK_PAGER is not set, **rwaddrcount** automatically invokes this program to display its output a screen at a time.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwaddrcount** may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **rwaddrcount** may use this environment variable. See the FILES section for details.

TZ

When the argument to the **--timestamp-format** switch includes `local` or when a SiLK installation is built to use the local timezone, the value of the TZ environment variable determines the timezone in which **rwaddrcount** displays timestamps. (If both of those are false, the TZ environment variable is ignored.) If the TZ environment variable is not set, the machine's default timezone is used. Setting TZ to the empty string or 0 causes timestamps to be displayed in UTC. For system information on the TZ variable, see **tzset(3)** or **environ(7)**. (To determine if SiLK was built with support for the local timezone, check the `Timezone support` value in the output of **rwaddrcount --version**.)

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

SEE ALSO

rwset(1), **rwsetcat(1)**, **rwstats(1)**, **rwtotal(1)**, **rwuniq(1)**, **silk(7)**, **tzset(3)**, **environ(7)**

NOTES

rwaddrcount only supports IPv4 addresses, and it will not be modified to support IPv6 addresses. To produce output similar to **rwaddrcount** for IPv6 addresses, use **rwuniq(1)**:

```
rwuniq --fields=sip --values=bytes,packets,records,stime,etime
```

When used in an IPv6 environment, **rwaddrcount** converts IPv6 flow records that contain addresses in the ::ffff:0:0/96 prefix to IPv4 and processes them. IPv6 records having addresses outside of that prefix are ignored.

rwaddrcount uses a fairly large hashtable to store data, but it is likely that as the amount of data expands, the application will take more time to process data.

Similar binning of records are produced by **rwstats(1)**, **rwtotal(1)**, and **rwuniq(1)**.

To generate a list of IP addresses without the volume information, use **rwset(1)**.

rwaggbag

Build a binary Aggregate Bag from SiLK Flow records

SYNOPSIS

```
rwaggbag --keys=KEY --counters=COUNTER
      [--note-strip] [--note-add=TEXT] [--note-file-add=FILE]
      [--invocation-strip] [--print-filenames] [--copy-input=PATH]
      [--compression-method=COMP_METHOD]
      [--ipv6-policy={ignore,asv4,mix,force,only}]
      [--output-path=PATH]
      [--site-config-file=FILENAME]
      {[--xargs] | [--xargs=FILENAME] | [FILE [FILE ...]]}


rwaggbag --help


rwaggbag --help-fields


rwaggbag --version
```

DESCRIPTION

rwaggbag reads SiLK Flow records and builds an Aggregate Bag file. To build an Aggregate Bag from textual input, use **rwaggbagbuild(1)**.

An *Aggregate Bag* is a binary file that maps a key to a counter, where the key and the counter are both composed of one or more fields. For example, an Aggregate Bag could contain the sum of the packet count and the sum of the byte count for each unique source IP and source port pair.

For each SiLK flow record **rwaggbag** reads, it extracts the values of the fields listed in the **--keys** switch, combines those fields into a key, searches for an existing bin that has that key and creates a new bin for that key if none is found, and adds the values for each of the fields listed in the **--counters** switch to the bin's counter. Both the **--keys** and **--counters** switches are required.

rwaggbag reads SiLK Flow records from the files named on the command line or from the standard input when no file names are specified and **--xargs** is not present. To read the standard input in addition to the named files, use `-` or `stdin` as a file name. If an input file name ends in `.gz`, the file is uncompressed as it is read. When the **--xargs** switch is provided, **rwaggbag** reads the names of the files to process from the named text file or from the standard input if no file name argument is provided to the switch. The input to **--xargs** must contain one file name per line.

If **rwaggbag** runs out of memory, it will exit immediately. The output Aggregate Bag file remains behind with a size of 0 bytes.

To print the contents of an Aggregate Bag as text, use **rwaggbagcat(1)**. The **rwaggbagbuild(1)** tool can create an Aggregate Bag from textual input. **rwaggbagtool(1)** allows you to manipulate binary Aggregate Bag files.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--keys=*KEY*

Create a key for binning flow records using the values of the comma-separated field(s) listed in *KEY*. The field names are case-insensitive, a name may be abbreviated to its shortest unique prefix, and a name may only be used one time. The list of available *KEY* fields are

sIPv4
: source IP address when IPv4

sIPv6
: source IP address when IPv6

dIPv4
: destination IP address when IPv4

dIPv6
: destination IP address when IPv6

sPort
: source port for TCP or UDP, or equivalent

dPort
: destination port for TCP or UDP, or equivalent

protocol
: IP protocol

packets
: count of packets recorded for this flow record

bytes
: count of bytes recorded for this flow record

flags
: bit-wise OR of TCP flags over all packets in the flow

sTime
: starting time of the flow, in seconds resolution

duration
: duration of the flow, in seconds resolution

eTime
: ending time of the flow, in seconds resolution

sensor
: numeric ID of the sensor where the flow was collected

input
: router SNMP input interface or vlanId if packing tools were configured to capture it (see **sensor.conf(5)**)

output
: router SNMP output interface or postVlanId

nhIPv4
router next hop IP address when IPv4

nhIPv6
router next hop IP address when IPv6

initialFlags
TCP flags on first packet in the flow as reported by **yaf(1)**

sessionFlags
bit-wise OR of TCP flags over all packets in the flow except the first as reported by **yaf**

attributes
flow attributes set by the flow generator

application
content of the flow as reported in the applabel field of **yaf**

class
class of the sensor at the collection point

type
type of the sensor at the collection point

icmpType
ICMP type value for ICMP and ICMPv6 flows, 0 otherwise

icmpCode
ICMP code value for ICMP and ICMPv6 flows, 0 otherwise

scc
the country code of the source IP address. Uses the mapping file specified by the SILK_COUNTRY_CODES environment variable or the *country_codes.pmap* mapping file, as described in FILES. (See also **ccfilter(3)**.) *Since SiLK 3.19.0.*

dcc
the country code of the destination IP address. See `scc`. *Since SiLK 3.19.0.*

--counters=*COUNTER*

Add to the bin determined by the fields in **--key** the values of the comma-separated field(s) listed in *COUNTER*. The field names are case-insensitive, a name may be abbreviated to its shortest unique prefix, and a name may only be used one time. The list of available *COUNTER* fields are

records
count of the number of flow records that match the key

sum-packets
the sum of the packet counts for flow records that match the key

sum-bytes
the sum of the byte counts for flow records that match the key

sum-duration
the sum of the durations (in seconds) for flow records that match the key

--note-strip

Do not copy the notes (annotations) from the input file(s) to the output file. When this switch is not specified, notes from the input file(s) are copied to the output.

--note-add=*TEXT*

Add the specified *TEXT* to the header of the output file as an annotation. This switch may be repeated to add multiple annotations to a file. To view the annotations, use the **rwfileinfo(1)** tool.

--note-file-add=*FILENAME*

Open *FILENAME* and add the contents of that file to the header of the output file as an annotation. This switch may be repeated to add multiple annotations. Currently the application makes no effort to ensure that *FILENAME* contains text; be careful that you do not attempt to add a SiLK data file as an annotation.

--invocation-strip

Do not record any command line history: do not copy the invocation history from the input files to the output file(s), and do not record the current command line invocation in the output. The invocation may be viewed with **rwfileinfo(1)**.

--print-filenames

Print to the standard error the names of input files as they are opened.

--copy-input=*PATH*

Copy all binary SiLK Flow records read as input to the specified file or named pipe. *PATH* may be `stdout` or `-` to write flows to the standard output as long as the **--output-path** switch is specified to redirect **rwaggbag**'s output to a different location.

--output-path=*PATH*

Write the binary Aggregate Bag output to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output. If *PATH* names an existing file, **rwaggbag** exits with an error unless the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. If this switch is not given, the output is written to the standard output. Attempting to write the binary output to a terminal causes **rwaggbag** to exit with an error.

--ipv6-policy=*POLICY*

Determine how IPv4 and IPv6 flows are handled when SiLK has been compiled with IPv6 support. When the switch is not provided, the SILK_IPV6_POLICY environment variable is checked for a policy. If it is also unset or contains an invalid policy, the *POLICY* is **mix**. When SiLK has not been compiled with IPv6 support, IPv6 flows are always ignored, regardless of the value passed to this switch or in the SILK_IPV6_POLICY variable. The supported values for *POLICY* are:

ignore

Ignore any flow record marked as IPv6, regardless of the IP addresses it contains. Only IP addresses contained in IPv4 flow records will be added to the Aggregate Bag.

asv4

Convert IPv6 flow records that contain addresses in the ::ffff:0:0/96 netblock (that is, IPv4-mapped IPv6 addresses) to IPv4 and ignore all other IPv6 flow records.

mix

Process the input as a mixture of IPv4 and IPv6 flow records. When creating a bag whose key is an IP address and the input contains IPv6 addresses outside of the ::ffff:0:0/96 netblock, this policy is equivalent to **force**; otherwise it is equivalent to **asv4**.

force

Convert IPv4 flow records to IPv6, mapping the IPv4 addresses into the ::ffff:0:0/96 netblock.

only

Process only flow records that are marked as IPv6. Only IP addresses contained in IPv6 flow records will be added to the Aggregate Bag.

--compression-method=*COMP_METHOD*

Specify the compression library to use when writing output files. If this switch is not given, the value in the SILK_COMPRESSION_METHOD environment variable is used if the value names an available compression method. When no compression method is specified, output to the standard output or to named pipes is not compressed, and output to files is compressed using the default chosen when SiLK was compiled. The valid values for *COMP_METHOD* are determined by which external libraries were found when SiLK was compiled. To see the available compression methods and the default method, use the **--help** or **--version** switch. SiLK can support the following *COMP_METHOD* values when the required libraries are available.

none

Do not compress the output using an external library.

zlib

Use the **zlib(3)** library for compressing the output, and always compress the output regardless of the destination. Using zlib produces the smallest output files at the cost of speed.

lzo1x

Use the *lzo1x* algorithm from the LZO real time compression library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead.

snappy

Use the *snappy* library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead. *Since SiLK 3.13.0.*

best

Use lzo1x if available, otherwise use snappy if available, otherwise use zlib if available. Only compress the output when writing to a file.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwaggbag** searches for the site configuration file in the locations specified in the FILES section.

--xargs

--xargs=*FILENAME*

Read the names of the input files from *FILENAME* or from the standard input if *FILENAME* is not provided. The input is expected to have one filename per line. **rwaggbag** opens each named file in turn and reads records from it as if the filenames had been listed on the command line.

--help

Print the available options and exit.

--help-fields

Print the names and descriptions of the keys and counters that may be used in the **--keys** and **--counters** switches and exit. *Since SiLK 3.22.0.*

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

To create an Aggregate Bag that sums the packet count for destination IPs addresses in the SiLK Flow file *data.rw*:

```
$ rwaggbag --key=dipv6 --counter=sum-packets data.rw   \
  | rwaggbagcat
```

To sum the number of records, packet count, and byte count for all flow records

```
$ rwaggbag --key=dport --counter=records,sum-packets,sum-bytes    \
       --output-path=dport.aggbag data.rw
```

To count the number of records seen for each unique source port, destination port, and protocol:

```
$ rwaggbag --key=sport,dport,proto --counter=records data.rw   \
  | rwaggbagcat
```

ENVIRONMENT

SILK_COUNTRY_CODES

This environment variable allows the user to specify the country code mapping file that **rwaggbag** uses when mapping an IP to a country for the `scc` and `dcc` keys. The value may be a complete path or a file relative to the SILK_PATH. See the FILES section for standard locations of this file.

SILK_IPV6_POLICY

This environment variable is used as the value for **--ipv6-policy** when that switch is not provided.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_COMPRESSION_METHOD

This environment variable is used as the value for **--compression-method** when that switch is not provided.

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwaggbag** may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **rwaggbag** may use this environment variable. See the FILES section for details.

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

$SILK_COUNTRY_CODES

$SILK_PATH/share/silk/country_codes.pmap

$SILK_PATH/share/country_codes.pmap

/usr/local/share/silk/country_codes.pmap

/usr/local/share/country_codes.pmap

Possible locations for the country code mapping file required by the `scc` and `dcc` keys.

NOTES

rwaggbag and the other Aggregate Bag tools were introduced in SiLK 3.15.0.

SEE ALSO

rwaggbagbuild(1), **rwaggbagcat(1)**, **rwaggbagtool(1)**, **rwbag(1)**, **rwfileinfo(1)**, **rwfilter(1)**, **rwnetmask(1)**, **rwset(1)**, **rwuniq(1)**, **ccfilter(3)**, **sensor.conf(5)**, **silk(7)**, **yaf(1)**, **zlib(3)**

rwaggbagbuild

Create a binary aggregate bag from non-flow data

SYNOPSIS

```
rwaggbagbuild [--fields=FIELDS]
      [--constant-field=FIELD=VALUE [--constant-field=FIELD=VALUE...]]
      [--column-separator=CHAR] [--no-titles]
      [--bad-input-lines=FILE] [--verbose] [--stop-on-error]
      [--note-add=TEXT] [--note-file-add=FILE]
      [--invocation-strip] [--compression-method=COMP_METHOD]
      [--output-path=PATH] [--site-config-file=FILENAME]
      {[--xargs] | [--xargs=FILENAME] | [FILE [FILE...]]}

rwaggbagbuild --help

rwaggbagbuild --help-fields

rwaggbagbuild --version
```

DESCRIPTION

rwaggbagbuild builds a binary Aggregate Bag file by reading one or more files containing textual input. To build an Aggregate Bag from SiLK Flow records, use **rwaggbag(1)**.

An *Aggregate Bag* is a binary file that maps a key to a counter, where the key and the counter are both composed of one or more fields. For example, an Aggregate Bag could contain the sum of the packet count and the sum of the byte count for each unique source IP and source port pair.

rwaggbagbuild reads its input from the files named on the command line or from the standard input when no file names are specified, when **--xargs** is not present, and when the standard input is not a terminal. To read the standard input in addition to the named files, use `-` or `stdin` as a file name. When the **--xargs** switch is provided, **rwaggbagbuild** reads the names of the files to process from the named text file or from the standard input if no file name argument is provided to the switch. The input to **--xargs** must contain one file name per line.

The new Aggregate Bag file is written to the location specified by the **--output-path** switch. If it is not provided, output is sent to the standard output when it is not connected to a terminal.

The Aggregate Bag file must have at least one field that it considers and key field and at least one field that it considers a counter field. See the description of the **--fields** switch.

In general (and as detailed below), each line of the text input files becomes one entry in the Aggregate Bag file. It is also possible to specify that each entry in the Aggregate Bag file contains additional fields, each with a specific value. These fields are specified by the **--constant-field** switch whose argument is a field name, an equals sign ('='), and a textual representation of a value. The named field becomes one of the key or counter fields in the Aggregate Bag file, and that field is given the specified value for each entry that is read from an input file. See the **--fields** switch in the OPTIONS section for the names of the fields and the acceptable forms of the textual input for each field.

The remainder of this section details how **rwaggbagbuild** processes each text input file to create an Aggregate Bag file.

When the **--fields** switch is specified, its argument specifies the key and counter fields that the new Aggregate Bag file is to contain. If **--fields** is not specified, the first line of the first input file is expected to contain field names, and those names determine the Aggregate Bag's key and counter. A field name of `ignore` causes **rwaggbagbuild** to ignore the values in that field when parsing the input.

The textual input is processed one line at a time. Comments begin with a '`#`'-character and continue to the end of the line; they are stripped from each line. After removing the comments, any line that is blank or contains only whitespace is ignored.

All other lines must contain valid input, which is a set of fields separated by a delimiter. The default delimiter is the virtual bar ('|') and may be changed with the **--column-separator** switch. Whitespace around a delimiter is allowed; however, using space or tab as the separator causes *each* space or tab character to be treated as a field delimiter. The newline character is not a valid delimiter character since it is used to denote records, and '`#`' is not a valid delimiter since it begins a comment.

The first line of each input file may contain delimiter-separated field names denoting in which order the fields appear in this input file. As mentioned above, when the **--fields** switch is not given, the first line of the first file determines the Aggregate Bag's key and counter. To tell **rwaggbagbuild** to treat the first line of each file as field values to be parsed, specify the **--no-titles** switch.

Every other line must contain delimiter-separated field values. A delimiter may follow the final field on a line. **rwaggbagbuild** ignores lines that contain either too few or too many fields.

See the description of the **--fields** switch in the OPTIONS section for the names of the fields and the acceptable forms of the textual input for each field.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--fields=*FIELDS*

Specify the fields in the input files. *FIELDS* is a comma separated list of field names. Field names are case-insensitive, and a name may be abbreviated to the shortest unique prefix. Other than the `ignore` field, a field name may not be specified more than once. The Aggregate Bag file must have at least one key field and at least one counter field.

The names of the fields that are considered key fields, their descriptions, and the format of the input that each expects are:

ignore
: field that **rwaggbagbuild** is to skip

sIPv4
: source IP address, IPv4 only; either the canonical dotted-quad format or an integer from 0 to 4294967295 inclusive

dIPv4
: destination IP address, IPv4 only; uses the same format as `sIPv4`

nhIPv4
: next hop IP address, IPv4 only; uses the same format as `sIPv4`

any-IPv4
: a generic IPv4 address; uses the same format as `sIPv4`

sIPv6
: source IP address, IPv6 only; the canonical hex-encoded format for IPv6 addresses

dIPv6
: destination IP address, IPv6 only; uses the same format as `sIPv6`

nhIPv6
: next hop IP address, IPv6 only; uses the same format as `sIPv6`

any-IPv6
: a generic IPv6 address; uses the same format as `sIPv6`

sPort
: source port; an integer from 0 to 65535 inclusive

dPort
: destination port; an integer from 0 to 65535 inclusive

any-port
: a generic port; an integer from 0 to 65535 inclusive

protocol
: IP protocol; an integer from 0 to 255 inclusive

packets
: packet count; an integer from 1 to 4294967295 inclusive

bytes
: byte count; an integer from 1 to 4294967295 inclusive

flags
: bit-wise OR of TCP flags over all packets; a string containing `F`, `S`, `R`, `P`, `A`, `U`, `E`, `C` in upper- or lowercase

initialFlags
: TCP flags on the first packet; uses the same form as `flags`

sessionFlags
: bit-wise OR of TCP flags on the second through final packet; uses the same form as `flags`

sTime
: starting time in seconds; uses the form `YYYY/MM/DD[:hh[:mm[:ss[.sss]]]]` (any milliseconds value is dropped). A `T` may be used in place of `:` to separate the day and hour fields. A floating point value between 536870912 and 2147483647 is also allowed and is treated as seconds since the UNIX epoch.

eTime
: ending time in seconds; uses the same format as `sTime`

any-time
: a generic time in seconds; uses the same format as `sTime`

duration
: duration of flow; a floating point value from 0.0 to 4294967.295

sensor
: sensor name or ID at the collection point; a string as given in **silk.conf(5)**

class
: class at collection point; a string as given in *silk.conf*

type
type at collection point; a string as given in *silk.conf*

input
router SNMP ingress interface or vlanId; an integer from 0 to 65535

output
router SNMP egress interface or postVlanId; an integer from 0 to 65535

any-snmp
a generic SNMP value; an integer from 0 to 65535

attribute
flow attributes set by the flow generator:

`S`
all the packets in this flow record are exactly the same size

`F`
flow generator saw additional packets in this flow following a packet with a FIN flag (excluding ACK packets)

`T`
flow generator prematurely created a record for a long-running connection due to a timeout or a byte-count threshold

`C`
flow generator created a record as a continuation of a previous record for a connection that exceeded a timeout or byte-count threshold

application
guess as to the content of the flow; as an integer from 0 to 65535

icmpType
ICMP type; an integer from 0 to 255 inclusive

icmpCode
ICMP code; an integer from 0 to 255 inclusive

scc
the country code of the source; accepts a two character string to use as the country of the source IP. The code is **not** checked for validity against the *country_codes.pmap* file. The code must be ASCII and it may contain two letters, a letter followed by a number, or the string `--`. *Since SiLK 3.19.0.*

dcc
the country code of the destination. See `scc`. *Since SiLK 3.19.0.*

any-cc
a generic country code. See `scc`. *Since SiLK 3.19.0.*

custom-key
a generic key; an integer from 0 to 4294967295 inclusive

The names and descriptions of the fields that are considered counter fields are listed next. For each, the type of input is an unsigned 64-bit number; that is, an integer from 0 to 18446744073709551615.

records
count of records that match the key

sum-packets
sum of packet counts

sum-bytes
: sum of byte counts

sum-duration
: sum of duration values

custom-counter
: a generic counter

--constant-field=*FIELD*=*VALUE*

For each entry (row) read from the input file(s), insert or replace a field named *FIELD* and set its value to *VALUE*. *VALUE* is a textual representation of the field's value as described in the description of the **--fields** switch above. When *FIELD* is a counter field and the same key appears multiple times in the input, *VALUE* is added to the counter multiple times. If a field named *FIELD* appears in an input file, its value from that file is ignored. Specify the **--constant-field** switch multiple times to insert multiple fields.

--column-separator=*CHAR*

When reading textual input, use the character *CHAR* as the delimiter between columns (fields) in the input. The default column separator is the vertical pipe ('|'). **rwaggbagbuild** normally ignores whitespace (space and tab) around the column separator; however, using space or tab as the separator causes *each* space or tab character to be treated as a field delimiter. The newline character is not a valid delimiter character since it is used to denote records, and '`#`' is not a valid delimiter since it begins a comment.

--bad-input-lines=*FILEPATH*

When parsing textual input, copy any lines than cannot be parsed to *FILEPATH*. The strings `stdout` and `stderr` may be used for the standard output and standard error, respectively. Each bad line is prepended by the name of the source input file, a colon, the line number, and a colon. On exit, **rwaggbagbuild** removes *FILEPATH* if all input lines were successfully parsed.

--verbose

When a textual input line fails to parse, print a message to the standard error describing the problem. When this switch is not specified, parsing failures are not reported. **rwaggbagbuild** continues to process the input after printing the message. To stop processing when a parsing error occurs, use **--stop-on-error**.

--stop-on-error

When a textual input line fails to parse, print a message to the standard error describing the problem and exit the program. When this occurs, the output file contains any records successfully created prior to reading the bad input line. The default behavior of **rwaggbagbuild** is to silently ignore parsing errors. To report parsing errors and continue processing the input, use **--verbose**.

--no-titles

Parse the first line of the input as field values. Normally when the **--fields** switch is specified, **rwaggbagbuild** examines the first line to determine if the line contains the names (titles) of fields and skips the line if it does. **rwaggbagbuild** exits with an error when **--no-titles** is given but **--fields** is not.

--note-add=*TEXT*

Add the specified *TEXT* to the header of the output file as an annotation. This switch may be repeated to add multiple annotations to a file. To view the annotations, use the **rwfileinfo(1)** tool.

--note-file-add=*FILENAME*

Open *FILENAME* and add the contents of that file to the header of the output file as an annotation. This switch may be repeated to add multiple annotations. Currently the application makes no effort to ensure that *FILENAME* contains text; be careful that you do not attempt to add a SiLK data file as an annotation.

--invocation-strip

Do not record the command used to create the Aggregate Bag file in the output. When this switch is not given, the invocation is written to the file's header, and the invocation may be viewed with **rwfileinfo(1)**.

--compression-method=*COMP_METHOD*

Specify the compression library to use when writing output files. If this switch is not given, the value in the SILK_COMPRESSION_METHOD environment variable is used if the value names an available compression method. When no compression method is specified, output to the standard output or to named pipes is not compressed, and output to files is compressed using the default chosen when SiLK was compiled. The valid values for *COMP_METHOD* are determined by which external libraries were found when SiLK was compiled. To see the available compression methods and the default method, use the **--help** or **--version** switch. SiLK can support the following *COMP_METHOD* values when the required libraries are available.

none

Do not compress the output using an external library.

zlib

Use the **zlib(3)** library for compressing the output, and always compress the output regardless of the destination. Using zlib produces the smallest output files at the cost of speed.

lzo1x

Use the *lzo1x* algorithm from the LZO real time compression library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead.

snappy

Use the *snappy* library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead.

best

Use lzo1x if available, otherwise use snappy if available, otherwise use zlib if available. Only compress the output when writing to a file.

--output-path=*PATH*

Write the binary Aggregate Bag output to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output. If *PATH* names an existing file, **rwaggbagbuild** exits with an error unless the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. If this switch is not given, the output is written to the standard output. Attempting to write the binary output to a terminal causes **rwaggbagbuild** to exit with an error.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwaggbagbuild** searches for the site configuration file in the locations specified in the FILES section.

--xargs

--xargs=*FILENAME*

Read the names of the input files from *FILENAME* or from the standard input if *FILENAME* is not provided. The input is expected to have one filename per line. **rwaggbagbuild** opens each named file in turn and reads text from it as if the filenames had been listed on the command line.

--help

Print the available options and exit.

--help-fields

Print the names and descriptions of the keys and counters that may be used in the **--fields** and **--constant-field** switches and exit. *Since SiLK 3.22.0.*

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

Assume the following textual data in the file *rec.txt*:

```
            dIP|dPort|   packets|     bytes|
  10.245.15.175|   80|       127|     12862|
192.168.251.186|29222|       131|    351213|
 10.247.186.130|   80|       596|     38941|
192.168.239.224|29362|       600|    404478|
192.168.215.219|   80|       400|     32375|
  10.255.252.19|28925|       404|   1052274|
192.168.255.249|   80|       112|      7412|
   10.208.7.238|29246|       109|    112977|
192.168.254.127|   80|       111|      9759|
  10.218.34.108|29700|       114|    461845|
```

To create an Aggregate Bag file from this data, provide the --fields switch with the names used by the Aggregate Bag tools:

```
$ rwaggbagbuild --fields=dipv4,dport,sum-packets,sum-bytes  \
      --output-path=ab.aggbag rec.txt
```

Use the **rwaggbagcat(1)** tool to view it:

```
$ rwaggbagcat ab.aggbag
          dIPv4|dPort|    sum-packets|           sum-bytes|
   10.208.7.238|29246|            109|              112977|
  10.218.34.108|29700|            114|              461845|
  10.245.15.175|   80|            127|               12862|
 10.247.186.130|   80|            596|               38941|
  10.255.252.19|28925|            404|             1052274|
```

```
192.168.215.219|   80|           400|             32375|
192.168.239.224|29362|           600|            404478|
192.168.251.186|29222|           131|            351213|
192.168.254.127|   80|           111|              9759|
192.168.255.249|   80|           112|              7412|
```

Create an Aggregate Bag from the destination port field and count the number of times each port appears, ignore all fields except the `dPort` fields and use **--constant-field** to add a new field:

```
$ rwaggbagbuild --fields=ignore,dport,ignore,ignore  \
       --constant-field=record=1                     \
  | rwaggbagcat
dPort|   records|
   80|         5|
28925|         1|
29222|         1|
29246|         1|
29362|         1|
29700|         1|
```

Alternatively, use **rwaggbagtool(1)** to get the same information from the *ab.aggbag* file created above:

```
$ rwaggbagtool --select-fields=dport        \
       --insert-field=record=1 ab.aggbag    \
  | rwaggbagcat
dPort|   records|
   80|         5|
28925|         1|
29222|         1|
29246|         1|
29362|         1|
29700|         1|
```

ENVIRONMENT

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_COMPRESSION_METHOD

This environment variable is used as the value for **--compression-method** when that switch is not provided.

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwaggbagbuild** may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **rwaggbagbuild** may use this environment variable. See the FILES section for details.

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

SEE ALSO

rwaggbag(1), **rwaggbagcat(1)**, **rwaggbagtool(1)**, **rwfileinfo(1)**, **rwset(1)**, **rwsetbuild(1)**, **rwsetcat(1)**, **rwsettool(1)**, **ccfilter(3)**, **silk.conf(5)**, **silk(7)**, **zlib(3)**

NOTES

rwaggbagbuild and the other Aggregate Bag tools were introduced in SiLK 3.15.0.

rwaggbagcat

Output a binary Aggregate Bag file as text

SYNOPSIS

```
rwaggbagcat [--fields=FIELDS
      [--missing-field=FIELD=STRING [--missing-field=FIELD=STRING...]]]
      [--timestamp-format=FORMAT] [--ip-format=FORMAT]
      [--integer-sensors] [--integer-tcp-flags]
      [--no-titles] [--no-columns] [--column-separator=C]
      [--no-final-delimiter] [{--delimited | --delimited=C}]
      [--output-path=PATH] [--pager=PAGER_PROG]
      [--site-config-file=FILENAME]
      [AGGBAGFILE [AGGBAGFILE...]]

rwaggbagcat --help

rwaggbagcat --help-fields

rwaggbagcat --version
```

DESCRIPTION

rwaggbagcat reads a binary Aggregate Bag as created by **rwaggbag(1)** or **rwaggbagbuild(1)**, converts it to text, and outputs it to the standard output, the pager, or the specified file.

As of SiLK 3.22.0, **rwaggbagcat** accepts a **--fields** switch to control the order in which the fields are printed.

rwaggbagcat reads the *AGGBAGFILE*s specified on the command line; if no *AGGBAGFILE* arguments are given, **rwaggbagcat** attempts to read an Aggregate Bag from the standard input. To read the standard input in addition to the named files, use `-` or `stdin` as an *AGGBAGFILE* name. If any input does not contain an Aggregate Bag file, **rwaggbagcat** prints an error to the standard error and exits abnormally.

When multiple *AGGBAGFILE*s are specified on the command line, each is handled individually. To process the files as a single Aggregate Bag, use **rwaggbagtool(1)** to combine the Aggregate Bags and pipe the output of **rwaggbagtool** into **rwaggbagcat**. Using **--fields** in this situation allows for a consistent output across the multiple files and causes the titles to appear only once. No value is printed if **--fields** names a key or counter that is not present in one of the files.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--fields=*FIELDS*

Print only the key and/or counter fields given in this comma separated list. Fields are printed in the order given in *FIELDS*, and keys and counters may appear in any order or not at all. Specifying

--fields only changes the order in which the columns are printed, it does not re-order the entries (rows) in the Aggregate Bag file. If *FIELDS* includes fields not present in an input Aggregate Bag file, prints the string specified for that field by **--missing-field** or an empty value. The title line is printed only one time even if multiple Aggregate Bag files are read.

The names of the fields that may appear in *FIELDS* are:

sIPv4
: source IP address, IPv4 only

dIPv4
: destination IP address, IPv4 only

nhIPv4
: next hop IP address, IPv4 only

any-IPv4
: a generic IPv4 address

sIPv6
: source IP address, IPv6 only

dIPv6
: destination IP address, IPv6 only

nhIPv6
: next hop IP address, IPv6 only

any-IPv6
: a generic IPv6 address

sPort
: source port

dPort
: destination port

any-port
: a generic port

protocol
: IP protocol

packets
: packet count

bytes
: byte count

flags
: bit-wise OR of TCP flags over all packets

initialFlags
: TCP flags on the first packet

sessionFlags
: bit-wise OR of TCP flags on the second through final packet

sTime
: starting time in seconds

eTime
: ending time in seconds

any-time
: a generic time in seconds

duration
: duration of flow

sensor
: sensor name or ID at the collection point

class
: class at collection point

type
: type at collection point

input
: router SNMP ingress interface or vlanId

output
: router SNMP egress interface or postVlanId

any-snmp
: a generic SNMP value

attribute
: flow attributes set by the flow generator

application
: guess as to the content of the flow

icmpType
: ICMP type

icmpCode
: ICMP code

scc
: the country code of the source

dcc
: the country code of the destination

any-cc
: a generic country code

custom-key
: a generic key

records
: counter: count of records that match the key

sum-packets
: counter: sum of packet counts

sum-bytes
: counter: sum of byte counts

sum-duration
: counter: sum of duration values

custom-counter
: counter: a generic counter

Since SiLK 3.22.0.

--missing-field=*FIELD=STRING*

When **--fields** is active, print *STRING* as the value for *FIELD* when *FIELD* is not present in the input Aggregate Bag file. The default value is the empty string. The switch may be repeated to set the missing value string for multiple fields. **rwaggbagcat** exits with an error if *FIELD* is not present in **--fields** or if this switch is specified but **--fields** is not. *STRING* may be any string. *Since SiLK 3.22.0.*

--timestamp-format=*FORMAT*

Specify the format, timezone, and/or modifier to use when printing timestamps. When this switch is not specified, the SILK_TIMESTAMP_FORMAT environment variable is checked for a format, timezone, and modifier. If it is empty or contains invalid values, timestamps are printed in the default format, and the timezone is UTC unless SiLK was compiled with local timezone support. *FORMAT* is a comma-separated list of a format, a timezone, and/or a modifier. The format is one of:

default
: Print the timestamps as *YYYY/MM/DD*T*hh*:*mm*:*ss.sss*.

iso
: Print the timestamps as *YYYY-MM-DD hh*:*mm*:*ss.sss*.

m/d/y
: Print the timestamps as *MM/DD/YYYY hh*:*mm*:*ss.sss*.

epoch
: Print the timestamps as the number of seconds since 00:00:00 UTC on 1970-01-01.

When a timezone is specified, it is used regardless of the default timezone support compiled into SiLK. The timezone is one of:

utc
: Use Coordinated Universal Time to print timestamps.

local
: Use the TZ environment variable or the local timezone.

--ip-format=*FORMAT*

Specify how IP addresses are printed, where *FORMAT* is a comma-separated list of the arguments described below. When this switch is not specified, the SILK_IP_FORMAT environment variable is checked for a value and that format is used if it is valid. The default *FORMAT* is `canonical`.

canonical
: Print IP addresses in the canonical format. If the column is IPv4, use dot-separated decimal (`192.0.2.1`). If the column is IPv6, use colon-separated hexadecimal (`2001:db8::1`) or a mixed IPv4-IPv6 representation for IPv4-mapped IPv6 addresses (the ::ffff:0:0/96 netblock, e.g., `::ffff:192.0.2.1`) and IPv4-compatible IPv6 addresses (the ::/96 netblock other than ::/127, e.g., `::192.0.2.1`).

no-mixed
: Print IP addresses in the canonical format (`192.0.2.1` or `2001:db8::1`) but do not used the mixed IPv4-IPv6 representations. For example, use `::ffff:c000:201` instead of `::ffff:192.0.2.1`. *Since SiLK 3.17.0.*

decimal

Print IP addresses as integers in decimal format. For example, print `192.0.2.1` and `2001:db8::1` as `3221225985` and `42540766411282592856903984951653826561`, respectively.

hexadecimal

Print IP addresses as integers in hexadecimal format. For example, print `192.0.2.1` and `2001:db8::1` as `c00000201` and `20010db800000000000000000000001`, respectively.

zero-padded

Make all IP address strings contain the same number of characters by padding numbers with leading zeros. For example, print `192.0.2.1` and `2001:db8::1` as `192.000.002.001` and `2001:0db8:0000:0000:0000:0000:0000:0001`, respectively. For IPv6 addresses, this setting implies `no-mixed`, so that `::ffff:192.0.2.1` is printed as `0000:0000:0000:0000:0000:ffff:c000:0201`. As of SiLK 3.17.0, may be combined with any of the above, including `decimal` and `hexadecimal`.

The following arguments modify certain IP addresses prior to printing. These arguments may be combined with the above formats.

map-v4

Change an IPv4 column to IPv4-mapped IPv6 addresses (addresses in the ::ffff:0:0/96 netblock) prior to formatting. *Since SiLK 3.17.0.*

unmap-v6

For an IPv6 column, change any IPv4-mapped IPv6 addresses (addresses in the ::ffff:0:0/96 netblock) to IPv4 addresses prior to formatting. *Since SiLK 3.17.0.*

The following argument is also available:

force-ipv6

Set *FORMAT* to `map-v4`,`no-mixed`.

--integer-sensors

Print the integer ID of the sensor rather than its name.

--integer-tcp-flags

Print the TCP flag fields (flags, initialFlags, sessionFlags) as an integer value. Typically, the characters `F,S,R,P,A,U,E,C` are used to represent the TCP flags.

--no-titles

Turn off column titles. By default, titles are printed.

--no-columns

Disable fixed-width columnar output.

--column-separator=*C*

Use specified character between columns and after the final column. When this switch is not specified, the default of '|' is used.

--no-final-delimiter

Do not print the column separator after the final column. Normally a delimiter is printed.

--delimited

--delimited=*C*

Run as if **--no-columns --no-final-delimiter --column-sep**=*C* had been specified. That is, disable fixed-width columnar output; if character *C* is provided, it is used as the delimiter between columns instead of the default '|'.

--output-path=*PATH*

Write the textual output to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output (and bypass the paging program). If *PATH* names an existing file, **rwaggbagcat** exits with an error unless the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. If this option is not given, the output is either sent to the pager or written to the standard output.

--pager=*PAGER_PROG*

When output is to a terminal, invoke the program *PAGER_PROG* to view the output one screen full at a time. This switch overrides the SILK_PAGER environment variable, which in turn overrides the PAGER variable. If the **--output-path** switch is given or if the value of the pager is determined to be the empty string, no paging is performed and all output is written to the terminal.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwaggbagcat** searches for the site configuration file in the locations specified in the FILES section.

--help

Print the available options and exit.

--help-fields

Print the names and descriptions of the keys and counters that may be used in the **--fields** and **--missing-field** switches and exit. *Since SiLK 3.22.0.*

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

The formatting switches on **rwaggbagcat** are similar to those on the other SiLK tools.

Creating and printing an Aggregate Bag file

First, use **rwaggbag(1)** to create an Aggregate Bag file from the SiLK Flow file *data.rw*:

```
$ rwaggbag --key=sport,dport --counter=sum-pack,sum-byte \
       --output-path=ab.aggbag data.rw
```

To print the contents of the Aggregate Bag file:

```
$ rwaggbagcat ab.aggbag | head -4
sPort|dPort|    sum-packets|           sum-bytes|
    0|    0|          73452|             6169968|
    0|  769|          15052|              842912|
    0|  771|          14176|              793856|
```

Reordering the columns

Use the **--fields** switch (added in SiLK 3.22.0) to control the order of the columns in the output or to select only some columns:

```
$ rwaggbagcat --fields=dPort,sPort,sum-bytes ab.aggbag | head -4
dPort|sPort|           sum-bytes|
    0|    0|             6169968|
  769|    0|              842912|
  771|    0|              793856|
```

The **--fields** switch only changes the positions of the columns. The `sPort` field is still the primary key in the output shown above.

The **--fields** switch may also include fields that are not in the input. By default, **rwaggbagcat** prints an empty value for those fields, but the **--missing-field** switch may be used to display any string instead. The argument to **--missing-field** is *FIELD*=*STRING* where *FIELD* is one of the fields in **--fields**.

```
$ rwaggbagcat --fields=sipv4,proto,dport,sum-bytes \
       --missing=sipv4=n/a ab.aggbag | head -4
         sIPv4|pro|dPort|           sum-bytes|
           n/a|   |    0|             6169968|
           n/a|   |  769|              842912|
           n/a|   |  771|              793856|
```

Using --fields with IP addresses

When creating an Aggregate Bag file with the source IP address and protocol as keys, **rwaggbagcat** prints the columns in a different order depending on whether the address is treated as IPv4 or IPv6.

When the key is the source IPv4 address and the protocol, the Aggregate Bag is built with the source address as the primary key:

```
$ rwaggbag --key=sipv4,proto --counter=records data.rw        \
  | rwaggbagcat
        sIPv4|pro|   records|
   10.4.52.235|  6|         1|
  10.5.231.251|  6|         1|
   10.9.77.117|  6|         1|
```

Reading the same file but treating the data as IPv6 results in the protocol being the primary key:

```
$ rwaggbag --key=sipv6,proto --counter=records data.rw        \
  | rwaggbagcat
```

```
pro|                                  sIPv6|   records|
  1|                    ::ffff:10.40.151.242|         1|
  1|                    ::ffff:10.44.140.138|         1|
  1|                     ::ffff:10.53.204.62|         1|
```

In the latter case, the **--fields** may be used to display the source IPv6 address first, but the switch only changes the positions of the columns, it does not reorder the entries (rows):

```
$ rwaggbag --key=sipv6,proto --counter=records data.rw         \
  | rwaggbagcat --fields=sipv6,proto,records
                                  sIPv6|pro|   records|
                   ::ffff:10.40.151.242|  1|         1|
                   ::ffff:10.44.140.138|  1|         1|
                    ::ffff:10.53.204.62|  1|         1|
```

Removing the columns or the title from the output

To produce comma separated data:

```
rwaggbagcat --delimited=, /tmp/ab.aggbag | head -4
sPort,dPort,sum-packets,sum-bytes
0,0,73452,6169968
0,769,15052,842912
0,771,14176,793856
```

To remove the title:

```
$ rwaggbagcat --no-title ab.aggbag | head -4
    0|    0|          73452|             6169968|
    0|  769|          15052|              842912|
    0|  771|          14176|              793856|
    0| 2048|          14356|             1205904|
```

Customizing the IP and timestamp format

To change the format of IP addresses:

```
$ rwaggbag --key=sipv4,dipv4 --counter=sum-pack,sum-byte data.rw   \
  | rwaggbagcat --ip-format=decimal | head -4
     sIPv4|     dIPv4|    sum-packets|           sum-bytes|
 168047851|3232295339|            255|               18260|
 168159227|3232293505|            331|              536169|
 168381813|3232282689|            563|               55386|
```

To change the format of timestamps:

```
$ rwaggbag --key=stime,etime --counter=sum-pack,sum-byte data.rw   \
  | rwaggbagcat --timestamp-format=epoch | head -4
```

```
      sTime|     eTime|    sum-packets|           sum-bytes|
1234396802|1234396802|              2|                 259|
1234396802|1234398594|            526|               38736|
1234396803|1234396803|              9|                 504|
```

ENVIRONMENT

SILK_IP_FORMAT

This environment variable is used as the value for **--ip-format** when that switch is not provided.

SILK_TIMESTAMP_FORMAT

This environment variable is used as the value for **--timestamp-format** when that switch is not provided.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_PAGER

When set to a non-empty string, **rwaggbagcat** automatically invokes this program to display its output a screen at a time. If set to an empty string, **rwaggbagcat** does not automatically page its output.

PAGER

When set and SILK_PAGER is not set, **rwaggbagcat** automatically invokes this program to display its output a screen at a time.

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwaggbagcat** may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files and plug-ins, **rwaggbagcat** may use this environment variable. See the FILES section for details.

TZ

When the argument to the **--timestamp-format** switch includes `local` or when a SiLK installation is built to use the local timezone, the value of the TZ environment variable determines the timezone in which **rwaggbagcat** displays timestamps. (If both of those are false, the TZ environment variable is ignored.) If the TZ environment variable is not set, the machine's default timezone is used. Setting TZ to the empty string or 0 causes timestamps to be displayed in UTC. For system information on the TZ variable, see **tzset(3)** or **environ(7)**. (To determine if SiLK was built with support for the local timezone, check the `Timezone support` value in the output of **rwaggbagcat --version**.)

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

NOTES

The **--fields**, **--missing-field**, and **--help-fields** switches were added in SiLK 3.22.0.

rwaggbagcat and the other Aggregate Bag tools were introduced in SiLK 3.15.0.

SEE ALSO

rwaggbag(1), **rwaggbagbuild(1)**, **rwaggbagtool(1)**, **silk(7)**, **tzset(3)**, **environ(7)**

rwaggbagtool

Manipulate binary Aggregate Bag files

SYNOPSIS

```
rwaggbagtool
      [{ --remove-fields=REMOVE_LIST | --select-fields=SELECT_LIST
         | --to-bag=BAG_KEY,BAG_COUNTER
         | --to-ipset=FIELD [--ipset-record-version=VERSION] }]
      [--insert-field=FIELD=VALUE [--insert-field=FIELD2=VALUE2...]]
      [{ --add | --subtract | --divide }]
      [--zero-divisor-result={error | remove | maximum | VALUE}]
      [--scalar-multiply={VALUE | FIELD=VALUE}
        [--scalar-multiply={VALUE | FIELD=VALUE}...]]
      [--min-field=FIELD=VALUE [--min-field=FIELD=VALUE...]]
      [--max-field=FIELD=VALUE [--max-field=FIELD=VALUE...]]
      [--set-intersect=FIELD=FILE [--set-intersect=FIELD=FILE...]]
      [--set-complement=FIELD=FILE [--set-complement=FIELD=FILE...]]
      [--output-path=PATH [--modify-inplace [--backup-path=BACKUP]]]
      [--note-strip] [--note-add=TEXT] [--note-file-add=FILE]
      [--compression-method=COMP_METHOD]
      [--site-config-file=FILENAME]
      [AGGBAG_FILE [AGGBAG_FILE ...]]

rwaggbagtool --help

rwaggbagtool --help-fields

rwaggbagtool --version
```

DESCRIPTION

rwaggbagtool performs operations on one or more Aggregate Bag files and creates a new Aggregate Bag file, a new Bag file, or an new IPset file. An *Aggregate Bag* is a binary file that maps a key to a counter, where the key and the counter are both composed of one or more fields. **rwaggbag(1)** and **rwaggbagbuild(1)** are the primary tools used to create an Aggregate Bag file. **rwaggbagcat(1)** prints a binary Aggregate Bag file as text.

The operations that **rwaggbagtool** supports are field manipulation (inserting or removing keys or counters), adding, subtracting, and dividing counters (all files must have the same keys and counters) across multiple Aggregate Bag files, multiplying all counters or only selected counters by a value, intersecting with an IPset, selecting rows based on minimum and maximum values of keys and counters, and creating a new IPset or Bag file.

rwaggbagtool processes the Aggregate Bag files listed on the command line. When no file names are specified, **rwaggbagtool** attempts to read an Aggregate Bag from the standard input. To read the standard input in addition to the named files, use `-` or `stdin` as a file name. If any input is not an Aggregate Bag file, **rwaggbagtool** prints an error to the standard error and exits with an error status.

By default, **rwaggbagtool**'s output is written to the standard output. Use **--output-path** to specify a different location. As of SiLK 3.21.0, **rwaggbagtool** supports the **--modify-inplace** switch which correctly handles the case when an input file is also used as the output file. That switch causes **rwaggbagtool** to write the output to a temporary file first and then replace the original output file. The **--backup-path** switch may be used in conjunction with **--modify-inplace** to set the pathname where the original output file is copied.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

The options are presented here in the order in which **rwaggbagtool** performs them: Field manipulation switches are applied to each file when it is read; multi-file operation switches combine the Aggregate Bags together; single-file operation switches are applied; filtering switches remove rows from the Aggregate Bag; the result is output as an Aggregate Bag, a standard Bag, or as an IPset.

Field manipulation switches

The following switches allow modification of the fields in the Aggregate Bag file. The **--remove-fields** and **--select-fields** switches are mutually exclusive, and they reduce the number of fields in the Aggregate Bag input files. Those switches also conflict with **--to-ipset** and **--to-bag** which resemble field selectors. The **--insert-field** switch is applied after **--remove-fields** or **--select-fields**, and it adds a field unless that field is already present.

--remove-fields=*REMOVE_LIST*

Remove the fields specified in *REMOVE_LIST* from each of the Aggregate Bag input files, where *REMOVE_LIST* is a comma-separated list of field names. This switch may include field names that are not in an Aggregate Bag input, and those field names are ignored. If a field name is included in this list and in a **--insert-field** switch, the field is given the value specified by the **--insert-field** switch, and the field is included in the output Aggregate Bag file. If removing a key field produces multiple copies of a key, the counters of those keys are merged. **rwaggbagbuild** exits with an error when this switch is used with **--select-fields**, **--to-ipset**, or **--to-bag**.

--select-fields=*SELECT_LIST*

For each Aggregate Bag input file, only use the fields in *SELECT_LIST*, a comma-separated list of field names. Alternatively, consider this switch as removing all fields that are not included in *SELECT_LIST*. This switch may include field names that are not in an Aggregate Bag input, and those field names are ignored. When a field name is included in this list and in a **--insert-field** switch, the field uses its value from the input Aggregate Bag file if present, and it uses the value specified in the **--insert-field** switch otherwise. If selecting only some key fields produces multiple copies of a key, the counters of those keys are merged. **rwaggbagbuild** exits with an error when this switch is used with **--remove-fields**, **--to-ipset**, or **--to-bag**.

--insert-field=*FIELD*=*VALUE*

For each entry read from an Aggregate Bag input file, insert a field named *FIELD* and set its value to *VALUE* if one of the following is true: (1)the input file does not contain a field named *FIELD* or (2)the input file does have a field named *FIELD* but it was removed by either (2a)being listed in the **--remove-fields** list or (2b)not being listed in the **--select-fields** list. That is, this switch only inserts

FIELD when *FIELD* is not present in the input Aggregate Bag, but specifying *FIELD* in **--remove-fields** removes it from the input. *VALUE* is a textual representation of the field's value as described in the description of the **--fields** switch in the **rwaggbagbuild(1)** tool. This switch may be repeated in order to insert multiple fields. If **--to-ipset** or **--to-bag** is specified, **--insert-field** may only name a field that is an argument to that switch.

Operations on multiple Aggregate Bag files

The following operations act on multiple Aggregate Bag files. These operations require all of the Aggregate Bag files to have the same set of key fields and counter fields. (Use the field manipulation switches to ensure this.) The values of the keys may differ, but the set of fields that comprise the key must match. It is an error if multiple operations are specified.

--add

Sum each of the counters for each key for all the Aggregate Bag input files. The keys in the result are the union of the set of keys that appear in all input files. Addition operations that overflow an unsigned 64-bit value are set to the maximum (18446744073709551615). If no other operation is specified, the add operation is the default.

--subtract

Subtract from the counters in the first Aggregate Bag file the counters in the second Aggregate Bag file, and repeat the process for each additional Aggregate Bag file. The keys in the result are a subset of the keys that appear in the first file: If a key does not appear in the first Aggregate Bag file, its counters are ignored in subsequent files. If a key does not appear in the second file, its counters in the first file are unchanged. Subtraction operations that result in a negative value are set to zero. If all counters for a key are zero, the key does not appear in the output.

--divide

Divide the counters in first Aggregate Bag file by the second Aggregate Bag file, and repeat the process for each additional Aggregate Bag file. The keys in the result are a subset of the keys that appear in the first file: If a key does not appear in the first Aggregate Bag file, its counters are ignored in subsequent files. If a key does not appear in the second file, its counters are treated as zero and the outcome is determined by the action specified by **--zero-divisor-result**. That option also determines the result when the two Aggregate Bag files have matching keys but a counter in the second bag is zero. If **--zero-divisor-result** is not given, **rwaggbagtool** exits with error if division by zero is detected. Since Aggregate Bag files do not support floating point numbers, the result of the division is rounded to the nearest integer (values ending in .5 are rounded up). *Since SiLK 3.22.0.*

While not an operation, the next switch is related to **--divide** and is described here.

--zero-divisor-result={ error | remove | maximum | *VALUE* }

Specify how to handle division by zero in the **--divide** operation, which can occur either because the first Aggregate Bag file (the dividend) contains a key that does not exist in the second file (the divisor) or because an individual counter in the divisor is zero. The supported arguments are:

error

Causes **rwaggbagtool** to exit with an error. This is the default when **--zero-divisor-result** is not given.

remove

Tells **rwaggbagtool** to remove this key from the output.

nochange
: Tells **rwaggbagtool** to leave the individual counter in the first Aggregate Bag unchanged.

maximum
: Sets the individual counter to the maximum value supported, which is the maximum unsigned 64-bit value (18446744073709551615).

VALUE
: Sets the individual counter to *VALUE*, which can be any unsigned 64-bit value (0 to 18446744073709551615 inclusive).

This switch has no effect when **--divide** is not used. *Since SiLK 3.22.0.*

Counter operations

The following switch modifies the counters in an Aggregate Bag file. The operation may be combined with any of those from the previous section. This operation occurs after the above and before any filtering operation.

--scalar-multiply=*VALUE*

--scalar-multiply=*FIELD*=*VALUE*

Multiply all counter fields or one counter field by a value. If the argument is a positive integer value (1 or greater), multiply all counters by that value. If the argument contains an equals sign, treat the part to the left as a counter's field name and the part to the right as the multiplier for that field: a non-negative integer value (0 or greater). The maximum *VALUE* is 18446744073709551615. This switch may be repeated; when a counter name is repeated or the all-counters form is repeated, the final multiplier is the **product** of all the values. *Since SiLK 3.22.0.*

Filtering switches

The following switches remove entries from the Aggregate Bag file based on a field's value. These switches are applied immediately before the output is generated.

--min-field=*FIELD*=*VALUE*

Remove from the Aggregate Bag file all entries where the value of the field *FIELD* is less than *VALUE*, where *VALUE* is a textual representation of the field's value as described in the description of the **--fields** switch in the **rwaggbagbuild(1)** tool. This switch is ignored if *FIELD* is not present in the Aggregate Bag. This switch may be repeated. *Since SiLK 3.17.0.*

--max-field=*FIELD*=*VALUE*

Remove from the Aggregate Bag file all entries where the value of the field *FIELD* is greater than *VALUE*, where *VALUE* is a textual representation of the field's value as described in the description of the **--fields** switch in the **rwaggbagbuild(1)** tool. This switch is ignored if *FIELD* is not present in the Aggregate Bag. This switch may be repeated. *Since SiLK 3.17.0.*

--set-intersect=*FIELD*=*SET_FILE*

Read an IPset from the stream *SET_FILE*, and remove from the Aggregate Bag file all entries where the value of the field *FIELD* is **not** present in the IPset. *SET_FILE* may be the name a file or the string `-` or `stdin` to read the IPset from the standard input. This switch is ignored if *FIELD* is not present in the Aggregate Bag. This switch may be repeated. *Since SiLK 3.17.0.*

--set-complement=*FIELD*=*SET_FILE*

Read an IPset from the stream *SET_FILE*, and remove from the Aggregate Bag file all entries where the value of the field *FIELD* is present in the IPset. *SET_FILE* may be the name a file or the string `-` or `stdin` to read the IPset from the standard input. This switch is ignored if *FIELD* is not present in the Aggregate Bag. This switch may be repeated. *Since SiLK 3.17.0.*

Output switches

The following switches control the output.

--to-bag=*BAG_KEY*,*BAG_COUNTER*

After operating on the Aggregate Bag input files, create a (normal) Bag file from the resulting Aggregate Bag. Use the *BAG_KEY* field as the key of the Bag, and the *BAG_COUNTER* field as the counter of the Bag. Write the Bag to the standard output or the destination specified by **--output-path**. When this switch is used, the only legal field names that may be used in the **--insert-field** switch are *BAG_KEY* and *BAG_COUNTER*. **rwaggbagbuild** exits with an error when this switch is used with **--remove-fields**, **--select-fields**, or **--to-ipset**.

--to-ipset=*FIELD*

After operating on the Aggregate Bag input files, create an IPset file from the resulting Aggregate Bag by treating the values in the field named *FIELD* as IP addresses, inserting the IP addresses into the IPset, and writing the IPset to the standard output or the destination specified by **--output-path**. When this switch is used, the only legal field name that may be used in the **--insert-field** switch is *FIELD*. **rwaggbagbuild** exits with an error when this switch is used with **--remove-fields**, **--select-fields**, or **--to-bag**.

--ipset-record-version=*VERSION*

Specify the format of the IPset records that are written to the output when the **--to-ipset** switch is used. *VERSION* may be 2, 3, 4, 5 or the special value 0. When the switch is not provided, the SILK_IPSET_RECORD_VERSION environment variable is checked for a version. The default version is 0.

0

Use the default version for an IPv4 IPset and an IPv6 IPset. Use the **--help** switch to see the versions used for your SiLK installation.

2

Create a file that may hold only IPv4 addresses and is readable by all versions of SiLK.

3

Create a file that may hold IPv4 or IPv6 addresses and is readable by SiLK 3.0 and later.

4

Create a file that may hold IPv4 or IPv6 addresses and is readable by SiLK 3.7 and later. These files are more compact that version 3 and often more compact than version 2.

5

Create a file that may hold only IPv6 addresses and is readable by SiLK 3.14 and later. When this version is specified, IPsets containing only IPv4 addresses are written in version 4. These files are usually more compact that version 4.

--output-path=*PATH*

Write the resulting Aggregate Bag, IPset (see **--to-ipset**), or Bag (see **--to-bag**) to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output. If *PATH* names an existing file, **rwaggbagtool** exits with an error unless the **--modify-inplace** switch is given or the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. If **--output-path** is not given, the output is written to the standard output. Attempting to write the binary output to a terminal causes **rwaggbagtool** to exit with an error.

--modify-inplace

Allow **rwaggbagtool** to overwrite an existing file and properly account for the output file (*PATH*) also being an input file. When this switch is given, **rwaggbagtool** writes the output to a temporary location first, then overwrites *PATH*. **rwaggbagtool** attempts to copy the permission, owner, and group from the original file to the new file. The switch is ignored when *PATH* does not exist or the output is the standard output or standard error. **rwaggbagtool** exits with an error when this switch is given and *PATH* is not a regular file. If **rwaggbagtool** encounters an error or is interrupted prior to closing the temporary file, the temporary file is removed. See also **--backup-path**. *Since SiLK 3.21.0.*

--backup-path=*BACKUP*

Move the file named by **--output-path** (*PATH*) to the path *BACKUP* immediately prior to moving the temporary file created by **--modify-inplace** over *PATH*. If *BACKUP* names a directory, the file is moved into that directory. This switch will overwrite an existing file. If *PATH* and *BACKUP* point to the same location, the output is written to *PATH* and no backup is created. If **BACKUP** cannot be created, the output is left in the temporary file and **rwaggbagtool** exits with a message and an error. **rwaggbagtool** exits with an error if this switch is given without **--modify-inplace**. *Since SiLK 3.21.0.*

--note-strip

Do not copy the notes (annotations) from the input files to the output file. Normally notes from the input files are copied to the output.

--note-add=*TEXT*

Add the specified *TEXT* to the header of the output file as an annotation. This switch may be repeated to add multiple annotations to a file. To view the annotations, use the **rwfileinfo(1)** tool.

--note-file-add=*FILENAME*

Open *FILENAME* and add the contents of that file to the header of the output file as an annotation. This switch may be repeated to add multiple annotations. Currently the application makes no effort to ensure that *FILENAME* contains text; be careful that you do not attempt to add a SiLK data file as an annotation.

--compression-method=*COMP_METHOD*

Specify the compression library to use when writing output files. If this switch is not given, the value in the SILK_COMPRESSION_METHOD environment variable is used if the value names an available compression method. When no compression method is specified, output to the standard output or to named pipes is not compressed, and output to files is compressed using the default chosen when SiLK was compiled. The valid values for *COMP_METHOD* are determined by which external libraries were found when SiLK was compiled. To see the available compression methods and the default method, use the **--help** or **--version** switch. SiLK can support the following *COMP_METHOD* values when the required libraries are available.

none
: Do not compress the output using an external library.

zlib
: Use the **zlib(3)** library for compressing the output, and always compress the output regardless of the destination. Using zlib produces the smallest output files at the cost of speed.

lzo1x
: Use the *lzo1x* algorithm from the LZO real time compression library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead.

snappy
: Use the *snappy* library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead.

best
: Use lzo1x if available, otherwise use snappy if available, otherwise use zlib if available. Only compress the output when writing to a file.

Miscellaneous switches

--site-config-file=*FILENAME*
: Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwaggbagtool** searches for the site configuration file in the locations specified in the FILES section.

--help
: Print the available options and exit.

--help-fields
: Print the names and descriptions of the fields that may be used in the command line options that require a field name. *Since SiLK 3.22.0.*

--version
: Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

Add two files

Read today's incoming flow records by type and use **rwaggbag(1)** to create an Aggregate Bag file for each, *in.aggbag* and *inweb.aggbag*, that count records using the protocol and both ports as the key. Add the counters in the two files to create *total.aggbag*. Use **rwaggbagcat(1)** to display the result.

```
$ rwfilter --type=in --all=-                                  \
  | rwaggbag --key=sport,dport,proto --counter=records        \
      --output-path=in.aggbag
```

```
$ rwfilter --type=inweb --all=-                              \
  | rwaggbag --key=sport,dport,proto --counter=records       \
       --output-path=inweb.aggbag
$ rwaggbagtool --add in.aggbag inweb.aggbag --output-path=total.aggbag
$ rwaggbagcat total.aggbag
```

Subtract a file

Subtract *inweb.aggbag* from *total.aggbag*.

```
$ rwaggbagtool --subtract total.aggbag inweb.aggbag    \
  | rwaggbagcat
```

Percent of traffic

Compute the percent of all incoming traffic per protocol and ports that was stored in the **inweb** type by multiplying the counters in *inweb.aggbag* by 100 and dividing by *total.aggbag*.

```
$ rwaggbagtool --scalar-multiply=100 inweb.aggbag  \
  | rwaggbagtool --divide stdin total.aggbag       \
  | rwaggbagcat
```

Create a file

Create an Aggregate Bag file from *data.rw* where the ports are the key and that sums the bytes and packets.

```
$ rwaggbag --key=sport,dport                         \
       --counter=sum-bytes,sum-packets data.rw       \
       --output-path=my-ab.aggbag
```

Choose selected fields

Using the previous file, get just the source port and byte count from the file *my-ab.aggbag*. One approach is to remove the destination port and packet count.

```
$ rwaggbagtool --remove=dport,sum-packets my-ab.aggbag  \
       --output-path=source-bytes.aggbag
```

The other approach selects the source port and byte count.

```
$ rwaggbagtool --select=sport,sum-bytes my-ag.aggbag    \
       --output-path=source-bytes.aggbag
```

To replace the packet count in *my-ab.aggbag* with zeros, remove the field and insert it with the value you want.

```
$ rwaggbagtool --remove=sum-packets --insert=sum-packets=0  \
       my-ab.aggbag --output-path=zero-packets.aggbag
```

Convert to different formats

To create a regular Bag with the source port and byte count from *my-ab.aggbag*, use the **--to-bag** switch:

```
$ rwaggbagtool --to-bag=sport,sum-bytes my-ab.aggbag  \
       --output-path=sport-byte.bag
```

The **--to-ipset** switch works similarly:

```
$ rwaggbag --key=sipv6,dipv6 --counter=records data-v6.rw  \
       --output-path=ips.aggbag
$ rwaggbagtool --to-ipset=dipv6 --output-path=dip.set
```

ENVIRONMENT

SILK_IPSET_RECORD_VERSION

This environment variable is used as the value for the **--ipset-record-version** when that switch is not provided.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_COMPRESSION_METHOD

This environment variable is used as the value for **--compression-method** when that switch is not provided.

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwaggbagtool** may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **rwaggbagtool** may use this environment variable. See the FILES section for details.

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

NOTES

The Aggregate Bag tools were added in SiLK 3.15.0.

SiLK 3.17.0 added the **--min-field**, **--max-field**, **--set-intersect**, and **--set-complement** switches.

Support for country codes was added in SiLK 3.19.0.

The **--modify-inplace** switch was added in SiLK 3.21. When **--backup-path** is also given, there is a small time window when the original file does not exist: the time between moving the original file to the backup location and moving the temporary file into place.

SEE ALSO

rwaggbag(1), **rwaggbagbuild(1)**, **rwaggbagcat(1)**, **rwfilter(1)**, **rwfileinfo(1)**, **silk(7)**, **zlib(3)**

rwappend

Append SiLK Flow file(s) to an existing SiLK Flow file

SYNOPSIS

```
rwappend [--create=[TEMPLATE_FILE]] [--print-statistics]
       [--site-config-file=FILENAME]
       TARGET_FILE SOURCE_FILE [SOURCE_FILE...]

rwappend --help

rwappend --version
```

DESCRIPTION

rwappend reads SiLK Flow records from the specified *SOURCE_FILE*s and appends them to the *TARGET_FILE*. If `stdin` is used as the name of one of the *SOURCE_FILE*s, SiLK flow records will be read from the standard input.

When the *TARGET_FILE* does not exist and the **--create** switch is not provided, **rwappend** will exit with an error. When **--create** is specified and *TARGET_FILE* does not exist, **rwappend** will create the *TARGET_FILE* using the same format, version, and byte-order as the specified *TEMPLATE_FILE*. If no *TEMPLATE_FILE* is given, the *TARGET_FILE* is created in the default format and version (the same format that **rwcat(1)** would produce).

The *TARGET_FILE* must be an actual file---it cannot be a named pipe or the standard output. In addition, the header of *TARGET_FILE* must not be compressed; that is, you cannot append to a file whose entire contents has been compressed with **gzip** (those files normally end in the `.gz` extension).

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--create

--create=*TEMPLATE_FILE*

Create the *TARGET_FILE* if it does not exist. The file will have the same format, version, and byte-order as the *TEMPLATE_FILE* if it is provided; otherwise the defaults are used. The *TEMPLATE_FILE* will **NOT** be appended to *TARGET_FILE* unless it also appears in as the name of a *SOURCE_FILE*.

--print-statistics

Print to the standard error the number of records read from each *SOURCE_FILE* and the total number of records appended to the *TARGET_FILE*.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwappend** searches for the site configuration file in the locations specified in the FILES section.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

Standard usage where the file to append to, *results.rw*, exists:

```
$ rwappend results.rw sample5.rw sample6.rw
```

To append files *sample*.rw* to *results.rw*, or to create *results.rw* using the same format as the first file argument (note that *sample1.rw* must be repeated):

```
$ rwappend results.rw --create=sample1.rw          \
       sample1.rw sample2.rw
```

If *results.rw* does not exist, the following two commands are equivalent:

```
$ rwappend --create results.rw sample1.rw sample2.rw
```

```
$ rwcat sample1.rw sample2.rw > results.rw
```

ENVIRONMENT

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwappend** may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **rwappend** may use this environment variable. See the FILES section for details.

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

SEE ALSO

rwcat(1), **silk(7)**

BUGS

When a *SOURCE_FILE* contains IPv6 flow records and the *TARGET_FILE* only supports IPv4 records, **rwappend** converts IPv6 records that contain addresses in the ::ffff:0:0/96 prefix to IPv4 and writes them to the *TARGET_FILE*. **rwappend** silently ignores IPv6 records having addresses outside of that prefix.

rwappend makes some attempts to avoid appending a file to itself (which would eventually exhaust the disk space) by comparing the names of files it is given; it should be smarter about this.

rwbag

Build a binary Bag from SiLK Flow records

SYNOPSIS

```
rwbag --bag-file=KEY,COUNTER,OUTPUTFILE
      [--bag-file=KEY,COUNTER,OUTPUTFILE ...]
      [{ --pmap-file=PATH | --pmap-file=MAPNAME:PATH }]
      [--note-strip] [--note-add=TEXT] [--note-file-add=FILE]
      [--invocation-strip] [--print-filenames] [--copy-input=PATH]
      [--compression-method=COMP_METHOD]
      [--ipv6-policy={ignore,asv4,mix,force,only}]
      [--site-config-file=FILENAME]
      {[--xargs] | [--xargs=FILENAME] | [FILE [FILE ...]]}

rwbag --help

rwbag --legacy-help

rwbag --version
```

LEGACY SYNOPSIS

```
rwbag [--sip-flows=OUTPUTFILE] [--dip-flows=OUTPUTFILE]
      [--sport-flows=OUTPUTFILE] [--dport-flows=OUTPUTFILE]
      [--proto-flows=OUTPUTFILE] [--sensor-flows=OUTPUTFILE]
      [--input-flows=OUTPUTFILE] [--output-flows=OUTPUTFILE]
      [--nhip-flows=OUTPUTFILE]
      [--sip-packets=OUTPUTFILE] [--dip-packets=OUTPUTFILE]
      [--sport-packets=OUTPUTFILE] [--dport-packets=OUTPUTFILE]
      [--proto-packets=OUTPUTFILE] [--sensor-packets=OUTPUTFILE]
      [--input-packets=OUTPUTFILE] [--output-packets=OUTPUTFILE]
      [--nhip-packets=OUTPUTFILE]
      [--sip-bytes=OUTPUTFILE] [--dip-bytes=OUTPUTFILE]
      [--sport-bytes=OUTPUTFILE] [--dport-bytes=OUTPUTFILE]
      [--proto-bytes=OUTPUTFILE] [--sensor-bytes=OUTPUTFILE]
      [--input-bytes=OUTPUTFILE] [--output-bytes=OUTPUTFILE]
      [--nhip-bytes=OUTPUTFILE]
      [--note-add=TEXT] [--note-file-add=FILE]
      [--print-filenames] [--copy-input=PATH]
      [--compression-method=COMP_METHOD]
      [--ipv6-policy={ignore,asv4,mix,force,only}]
      [--site-config-file=FILENAME]
      {[--xargs] | [--xargs=FILENAME] | [FILE [FILE ...]]}
```

DESCRIPTION

rwbag reads SiLK Flow records and builds one or more Bag files. A Bag is similar to a set but each key is associated with a counter. Usually the key is some aspect of a flow record (an IP address, a port, the

protocol, et cetera), and the counter is a volume (such as the number of flow records or the sum or bytes or packets) for the flow records that match that key. A Bag file supports a single key field and a single counter field; use the Aggregate Bag tools (e.g., **rwaggbag(1)**) when the key or counter contains multiple fields.

The **--bag-file** switch is required and it specifies how to create a Bag file. The argument to the switch names the key field to use for the bag, the counter field, and the location where the bag file is to be written. The switch may be repeated to create multiple Bag files.

rwbag reads SiLK Flow records from the files named on the command line or from the standard input when no file names are specified and **--xargs** is not present. To read the standard input in addition to the named files, use `-` or `stdin` as a file name. If an input file name ends in `.gz`, the file is uncompressed as it is read. When the **--xargs** switch is provided, **rwbag** reads the names of the files to process from the named text file or from the standard input if no file name argument is provided to the switch. The input to **--xargs** must contain one file name per line.

If adding a value to a key would cause the value to overflow the maximum value that Bags support, the key's value will be set to the maximum and processing will continue. In addition, if this is the first value to overflow in this Bag, a warning will be printed to the standard error.

If **rwbag** runs out of memory, it will exit immediately. The output Bag files will remain behind, each with a size of 0 bytes.

Use **rwbagcat(1)** to see the contents of a bag. To create a bag from textual input or from an IPset, use **rwbagbuild(1)**. **rwbagtool(1)** allows you to manipulate binary bag files.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--bag-file=*KEY*,*COUNTER*,*OUTPUTFILE*

Bin flow records by unique *KEY*, compute the *COUNTER* for each bin, and write the result to *OUTPUTFILE*. The list of available *KEY* and *COUNTER* values are given immediately below. *OUTPUTFILE* is the name of a non-existent file, a named pipe, or the keyword `stdout` or `-` to write the binary Bag to the standard output. Repeat the **--bag-file** switch to create multiple Bag files in a single pass over the data. Only one *OUTPUTFILE* may use the standard output. See LEGACY BAG CREATION SWITCHES for deprecated methods to create Bag files. This switch or one of legacy equivalents is required. *Since SiLK 3.12.0.*

rwbag supports the following names for *KEY*. The case of *KEY* is ignored.

sIPv4
: source IP address, either IPv4 or IPv6

sIPv6
: source IP address, either IPv4 or IPv6

dIPv4
: destination IP address, either IPv4 or IPv6

dIPv6
: destination IP address, either IPv4 or IPv6

sPort
: source port for TCP or UDP, or equivalent

dPort
: destination port for TCP or UDP, or equivalent

protocol
: IP protocol

packets
: count of packets recorded for this flow record

bytes
: count of bytes recorded for this flow record

flags
: bit-wise OR of TCP flags over all packets in the flow

sTime
: starting time of the flow, in seconds resolution

duration
: duration of the flow, in seconds resolution

eTime
: ending time of the flow, in seconds resolution

sensor
: numeric ID of the sensor where the flow was collected

input
: router SNMP input interface or vlanId if packing tools were configured to capture it (see **sensor.conf(5)**)

output
: router SNMP output interface or postVlanId

nhIPv4
: router next hop IP address, either IPv4 or IPv6

nhIPv6
: router next hop IP address, either IPv4 or IPv6

initialFlags
: TCP flags on first packet in the flow

sessionFlags
: bit-wise OR of TCP flags over all packets except the first in the flow

attributes
: flow attributes set by the flow generator

application
: guess as to the content of the flow

sip-country
: the country code of the source IP address. Uses the mapping file specified by the SILK_COUNTRY_CODES environment variable or the *country_codes.pmap* mapping file, as described in FILES. (See also **ccfilter(3)**.) *Since SiLK 3.12.0.*

scc
: an alias for sip-country

dip-country
: the country code of the destination IP address

dcc
an alias for dip-country

sip-pmap:MAPNAME
the value that the source IP address maps to in the mapping file whose map-name is MAPNAME. The type of that prefix map must be IPv4-address or IPv6-address. Use **--pmap-file** to load the mapping file and optionally set its map-name. Since the MAPNAME must be known when the **--bag-file** switch is parsed, the **--pmap-file** switch(es) should precede the **--bag-file** switch(es).

dip-pmap:MAPNAME
the value that the destination IP address maps to in the mapping file whose map-name is MAPNAME. See `sip-pmap:MAPNAME`.

sport-pmap:MAPNAME
the value that the protocol/source-port pair maps to in the mapping file whose map-name is MAPNAME. The type of that prefix map must be proto-port. Use **--pmap-file** to load the mapping file and optionally set its map-name. Since the MAPNAME must be known when the **--bag-file** switch is parsed, the **--pmap-file** switch(es) should precede the **--bag-file** switch(es).

dport-pmap:MAPNAME
the value that the protocol/destination-port pair maps to in the mapping file whose map-name is MAPNAME. See `sport-pmap:MAPNAME`.

rwbag supports the following names for *COUNTER*. The case of *COUNTER* is ignored.

records
count of the number of flow records that match the key

flows
an alias for records

sum-packets
the sum of the packet counts for flow records that match the key

packets
an alias for sum-packets

sum-bytes
the sum of the byte counts for flow records that match the key

bytes
an alias for sum-bytes

--pmap-file=*PATH*

--pmap-file=*MAPNAME*:*PATH*

Load the the prefix map file from *PATH* for use when the key part of the argument to the **--bag-file** switch is one of `sip-pmap`, `dip-pmap`, `sport-pmap`, or `dport-pmap`. Specify *PATH* as `-` or `stdin` to read from the standard input. If *MAPNAME* is specified, it overrides the map-name contained in the prefix map file itself. If no map-name is available, **rwbag** exits with an error. The switch may be repeated to load multiple prefix map files; each file must have a unique map-name. To create a prefix map file, use **rwpmapbuild(1)**. *Since SiLK 3.12.0.*

--note-strip

Do not copy the notes (annotations) from the input files to the output file(s). When this switch is not specified, notes from the input files are copied to the output. *Since SiLK 3.12.2.*

--note-add=*TEXT*

Add the specified *TEXT* to the header of every output file as an annotation. This switch may be repeated to add multiple annotations to a file. To view the annotations, use the **rwfileinfo(1)** tool.

--note-file-add=*FILENAME*

Open *FILENAME* and add the contents of that file to the header of every output file as an annotation. This switch may be repeated to add multiple annotations. Currently the application makes no effort to ensure that *FILENAME* contains text; be careful that you do not attempt to add a SiLK data file as an annotation.

--invocation-strip

Do not record any command line history: do not copy the invocation history from the input files to the output file(s), and do not record the current command line invocation in the output. The invocation may be viewed with **rwfileinfo(1)**. *Since SiLK 3.12.0.*

--print-filenames

Print to the standard error the names of input files as they are opened.

--copy-input=*PATH*

Copy all binary SiLK Flow records read as input to the specified file or named pipe. *PATH* may be `stdout` or `-` to write flows to the standard output as long as no Bag file is being written there.

--ipv6-policy=*POLICY*

Determine how IPv4 and IPv6 flows are handled when SiLK has been compiled with IPv6 support. When the switch is not provided, the SILK_IPV6_POLICY environment variable is checked for a policy. If it is also unset or contains an invalid policy, the *POLICY* is **mix**. When SiLK has not been compiled with IPv6 support, IPv6 flows are always ignored, regardless of the value passed to this switch or in the SILK_IPV6_POLICY variable. The supported values for *POLICY* are:

ignore

Ignore any flow record marked as IPv6, regardless of the IP addresses it contains. Only IP addresses contained in IPv4 flow records will be added to the bag(s).

asv4

Convert IPv6 flow records that contain addresses in the ::ffff:0:0/96 netblock (that is, IPv4-mapped IPv6 addresses) to IPv4 and ignore all other IPv6 flow records.

mix

Process the input as a mixture of IPv4 and IPv6 flow records. When creating a bag whose key is an IP address and the input contains IPv6 addresses outside of the ::ffff:0:0/96 netblock, this policy is equivalent to **force**; otherwise it is equivalent to **asv4**.

force

Convert IPv4 flow records to IPv6, mapping the IPv4 addresses into the ::ffff:0:0/96 netblock.

only

Process only flow records that are marked as IPv6. Only IP addresses contained in IPv6 flow records will be added to the bag(s).

Regardless of the IPv6 policy, when all IPv6 addresses in the bag are in the ::ffff:0:0/96 netblock, **rwbag** treats them as IPv4 addresses and writes an IPv4 bag. When any other IPv6 addresses are present in the bag, the IPv4 addresses in the bag are mapped into the ::ffff:0:0/96 netblock and **rwbag** writes an IPv6 bag.

--compression-method=*COMP_METHOD*

Specify the compression library to use when writing output files. If this switch is not given, the value in the SILK_COMPRESSION_METHOD environment variable is used if the value names an available compression method. When no compression method is specified, output to the standard output or to named pipes is not compressed, and output to files is compressed using the default chosen when SiLK was compiled. The valid values for *COMP_METHOD* are determined by which external libraries were found when SiLK was compiled. To see the available compression methods and the default method, use the **--help** or **--version** switch. SiLK can support the following *COMP_METHOD* values when the required libraries are available.

none

Do not compress the output using an external library.

zlib

Use the **zlib(3)** library for compressing the output, and always compress the output regardless of the destination. Using zlib produces the smallest output files at the cost of speed.

lzo1x

Use the *lzo1x* algorithm from the LZO real time compression library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead.

snappy

Use the *snappy* library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead. *Since SiLK 3.13.0.*

best

Use lzo1x if available, otherwise use snappy if available, otherwise use zlib if available. Only compress the output when writing to a file.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwbag** searches for the site configuration file in the locations specified in the FILES section.

--xargs

--xargs=*FILENAME*

Read the names of the input files from *FILENAME* or from the standard input if *FILENAME* is not provided. The input is expected to have one filename per line. **rwbag** opens each named file in turn and reads records from it as if the filenames had been listed on the command line.

--help

Print the available options and exit.

--legacy-help

Print help, including legacy switches. See the LEGACY BAG CREATION SWITCHES section below for these switches.

--version

Print the version number and information about how SiLK was configured, then exit the application.

LEGACY BAG CREATION SWITCHES

The following switches are deprecated as of SiLK 3.12.0. These switches may be used in conjunction with the **--bag-file** switch.

--sip-flows=*OUTPUTFILE*

Equivalent to **--bag-file**=sIPv4,records,*OUTPUTFILE*. Count number of flows by unique source IP.

--sip-packets=*OUTPUTFILE*

Equivalent to **--bag-file**=sIPv4,sum-packets,*OUTPUTFILE*. Count number of packets by unique source IP.

--sip-bytes=*OUTPUTFILE*

Equivalent to **--bag-file**=sIPv4,sum-bytes,*OUTPUTFILE*. Count number of bytes by unique source IP.

--dip-flows=*OUTPUTFILE*

Equivalent to **--bag-file**=dIPv4,records,*OUTPUTFILE*. Count number of flows by unique destination IP.

--dip-packets=*OUTPUTFILE*

Equivalent to **--bag-file**=dIPv4,sum-packets,*OUTPUTFILE*. Count number of packets by unique destination IP.

--dip-bytes=*OUTPUTFILE*

Equivalent to **--bag-file**=dIPv4,sum-bytes,*OUTPUTFILE*. Count number of bytes by unique destination IP.

--sport-flows=*OUTPUTFILE*

Equivalent to **--bag-file**=sPort,records,*OUTPUTFILE*. Count number of flows by unique source port.

--sport-packets=*OUTPUTFILE*

Equivalent to **--bag-file**=sPort,sum-packets,*OUTPUTFILE*. Count number of packets by unique source port.

--sport-bytes=*OUTPUTFILE*

Equivalent to **--bag-file**=sPort,sum-bytes,*OUTPUTFILE*. Count number of bytes by unique source port.

--dport-flows=*OUTPUTFILE*

Equivalent to **--bag-file**=dPort,records,*OUTPUTFILE*. Count number of flows by unique destination port.

--dport-packets=*OUTPUTFILE*

Equivalent to **--bag-file**=dPort,sum-packets,*OUTPUTFILE*. Count number of packets by unique destination port.

--dport-bytes=*OUTPUTFILE*

Equivalent to **--bag-file**=dPort,sum-bytes,*OUTPUTFILE*. Count number of bytes by unique destination port.

--proto-flows=*OUTPUTFILE*

Equivalent to **--bag-file**=protocol,records,*OUTPUTFILE*. Count number of flows by unique protocol.

--proto-packets=*OUTPUTFILE*

Equivalent to **--bag-file**=protocol,sum-packets,*OUTPUTFILE*. Count number of packets by unique protocol.

--proto-bytes=*OUTPUTFILE*

Equivalent to **--bag-file**=protocol,sum-bytes,*OUTPUTFILE*. Count number of bytes by unique protocol.

--sensor-flows=*OUTPUTFILE*

Equivalent to **--bag-file**=sensor,records,*OUTPUTFILE*. Count number of flows by unique sensor ID.

--sensor-packets=*OUTPUTFILE*

Equivalent to **--bag-file**=sensor,sum-packets,*OUTPUTFILE*. Count number of packets by unique sensor ID.

--sensor-bytes=*OUTPUTFILE*

Equivalent to **--bag-file**=sensor,sum-bytes,*OUTPUTFILE*. Count number of bytes by unique sensor ID.

--input-flows=*OUTPUTFILE*

Equivalent to **--bag-file**=input,records,*OUTPUTFILE*. Count number of flows by unique input interface index.

--input-packets=*OUTPUTFILE*

Equivalent to **--bag-file**=input,sum-packets,*OUTPUTFILE*. Count number of packets by unique input interface index.

--input-bytes=*OUTPUTFILE*

Equivalent to **--bag-file**=input,sum-bytes,*OUTPUTFILE*. Count number of bytes by unique input interface index.

--output-flows=*OUTPUTFILE*

Equivalent to **--bag-file**=output,records,*OUTPUTFILE*. Count number of flows by unique output interface index.

--output-packets=*OUTPUTFILE*

Equivalent to **--bag-file**=output,sum-packets,*OUTPUTFILE*. Count number of packets by unique output interface index.

--output-bytes=*OUTPUTFILE*

Equivalent to **--bag-file**=output,sum-bytes,*OUTPUTFILE*. Count number of bytes by unique output interface index.

--nhip-flows=*OUTPUTFILE*

Equivalent to **--bag-file**=nhIPv4,records,*OUTPUTFILE*. Count number of flows by unique next hop IP.

--nhip-packets=*OUTPUTFILE*

Equivalent to **--bag-file**=nhIPv4,sum-packets,*OUTPUTFILE*. Count number of packets by unique next hop IP.

--nhip-bytes=*OUTPUTFILE*

Equivalent to **--bag-file**=nhIPv4,sum-bytes,*OUTPUTFILE*. Count number of bytes by unique next hop IP.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

Bag of Protocol:Byte

Read the SiLK Flow file *data.rw* and create the Bag *proto-byte.bag* that contains the total byte-count seen for each protocol by using protocol as the key and sum-bytes as the counter:

```
$ rwbag --bag-file=protocol,sum-bytes,proto-byte.bag data.rw
```

Use **rwbagcat(1)** to view the result:

```
$ rwbagcat proto-byte.bag
          1|            10695328|
          6|        120536195111|
         17|            24500079|
```

Specify the output path as `-` to pass the Bag file from **rwbag** directly into **rwbagcat**.

```
$ rwbag --bag-file=protocol,sum-bytes,- data.rw    \
  | rwbagcat
          1|            10695328|
          6|        120536195111|
         17|            24500079|
```

Compare that to this **rwuniq(1)** command.

```
$ rwuniq --field=protocol --value=bytes --sort-output data.rw
pro|               Bytes|
  1|            10695328|
  6|        120536195111|
 17|            24500079|
```

One advantage of Bag files over **rwuniq** is that the data remains in binary form where it can be manipulated by **rwbagtool(1)**.

Two Bags in a Single Pass

Read records from **rwfilter(1)** and build Bag files *sip-flow.bag* and *dip-flow.bag* that count the number of flows seen for each source address and for each destination address, respectively.

```
$ rwfilter ... --pass=stdout                          \
  | rwbag --bag-file=sipv4,records,sip-flow.bag       \
       --bag-file=dipv4,records,dip-flow.bag
```

Using a Network Prefix

To create *sip16-byte.bag* that contains the number of bytes seen for each /16 found in the source address field, use the **rwnetmask(1)** tool prior to feeding the input to **rwbag**:

```
$ rwfilter ... --pass=stdout                          \
  | rwnetmask --4sip-prefix-length=16                 \
  | rwbag --bag-file=sipv4,sum-bytes,sip16-byte.bag

$ rwbagcat sip16-byte.bag | head -4
       10.4.0.0|               18260|
       10.5.0.0|              536169|
       10.9.0.0|               55386|
      10.11.0.0|             5110438|
```

To print the IP addresses of an existing Bag into /16 prefixes, use the **--network-structure** switch of **rwbagcat(1)**.

```
$ rwfilter ... --pass=stdout                      \
  | rwbag --bag-file=sipv4,sum-bytes,-            \
  | rwbagcat --network-structure=B                \
  | head -4
       10.4.0.0/16|               18260|
       10.5.0.0/16|              536169|
       10.9.0.0/16|               55386|
      10.11.0.0/16|             5110438|
```

Bag of Country Codes

As of SiLK 3.12.0, a Bag file may contain a country code as its key. Create *scc-pkt.bag* that sums the packet count by country.

```
$ rwbag --bag-file=sip-country,sum-packets,scc-pkt.bag
$ rwbagcat scc-pkt.bag
--|                 840|
a1|                 284|
a2|                   1|
ae|                   8|
```

Bag of Prefix Map Values

rwbag and **rwbagbuild(1)** can use a prefix map file as the key in a Bag file as of SiLK 3.12.0. For example, to lookup each source address in the prefix map file *ip-map.pmap* that maps from address to "type of service", use the **--pmap-file** switch to specify the prefix map file, and specify the Bag's key as `sip-pmap:`*map-name*, where *map-name* is either the map-name stored in the prefix map file or a name that is provided as part of the **--pmap-file** argument. (A prefix map's map-name is available via the **rwfileinfo(1)** command.)

```
$ rwfileinfo --field=prefix-map ip-map.pmap
ip-map.pmap:
  prefix-map          v1: service-host
$
$ rwbag --pmap-file=ip-map.pmap                              \
       --bag-file=sip-pmap:service-host,bytes,srvhost.bag  \
       data.rw
```

Multiple **--pmap-file** switches may be specified which may be useful when generating multiple Bag files in a single invocation. On the command line, the **--pmap-file** switch that defines the map-name must preceded the **--bag-file** where the map-name is used.

The prefix map file is not stored as part of the Bag, so you must provide the name of the prefix map when running **rwbagcat**.

```
$ rwbagcat srvhost.bag
rwbagcat: The --pmap-file switch is required for \
        Bags containing sip-pmap keys
$ rwbagcat --pmap-file=ip-map.pmap srvhost.bag
         external|         59950837766|
         internal|         60602999159|
              ntp|              588316|
              dns|            14404581|
             dhcp|             2560696|
```

rwbag also has support for prefix map files that map from a protocol-port pair to a label. The *proto-port.pmap* file does not have a map-name so a name must be provided on the **rwbag** command line.

```
$ rwfileinfo --field=prefix-map proto-port.pmap
proto-port.pmap:
$
$ rwbag --pmap-file=srvport:proto-port.pmap                \
       --bag-file=sip-pmap:srvport,flows,srvport.bag       \
       data.rw
$ rwbagcat --pmap-file=proto-port.pmap srvport.bag | head -4
     ICMP|               15622|
      UDP|               62216|
  UDP/DNS|               62216|
 UDP/DHCP|               15614|
```

ENVIRONMENT

SILK_COUNTRY_CODES

This environment variable allows the user to specify the country code mapping file that **rwbag** uses when mapping an IP to a country for the `sip-country` and `dip-country` keys. The value may be a complete path or a file relative to the SILK_PATH. See the FILES section for standard locations of this file.

SILK_IPV6_POLICY

This environment variable is used as the value for **--ipv6-policy** when that switch is not provided.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_COMPRESSION_METHOD

This environment variable is used as the value for **--compression-method** when that switch is not provided. *Since SiLK 3.13.0.*

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwbag** may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **rwbag** may use this environment variable. See the FILES section for details.

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

$SILK_COUNTRY_CODES

$SILK_PATH/share/silk/country_codes.pmap

$SILK_PATH/share/country_codes.pmap

/usr/local/share/silk/country_codes.pmap

/usr/local/share/country_codes.pmap

Possible locations for the country code mapping file required by the `sip-country` and `dip-country` keys.

SEE ALSO

rwbagbuild(1), **rwbagcat(1)**, **rwbagtool(1)**, **rwaggbag(1)**, **rwfileinfo(1)**, **rwfilter(1)**, **rwnetmask(1)**, **rwpmapbuild(1)**, **rwuniq(1)**, **ccfilter(3)**, **sensor.conf(5)**, **silk(7)**, **zlib(3)**

rwbagbuild

Create a binary Bag from non-flow data

SYNOPSIS

```
rwbagbuild { --set-input=SETFILE | --bag-input=TEXTFILE }
      [--delimiter=C] [--proto-port-delimiter=C]
      [--default-count=DEFAULTCOUNT]
      [--key-type=FIELD_TYPE] [--counter-type=FIELD_TYPE]
      [{ --pmap-file=PATH | --pmap-file=MAPNAME:PATH }]
      [--note-add=TEXT] [--note-file-add=FILE]
      [--invocation-strip] [--compression-method=COMP_METHOD]
      [--output-path=PATH]


rwbagbuild --help


rwbagbuild --version
```

DESCRIPTION

rwbagbuild builds a binary Bag file from an IPset file or from textual input. A Bag is a set of keys where each key is associated with a counter. Usually the key is some aspect of a flow record (an IP address, a port, the protocol, et cetera), and the counter is a volume (such as the number of flow records or the sum or bytes or packets) for the flow records that match that key.

Either **--set-input** or **--bag-input** must be provided to specify the type and the location of the input file. To read from the standard input, specify `stdin` or `-` as the argument to the switch.

Each occurrence of a unique key adds a counter value to the Bag file for that key, where the counter is the value specified by **--default-count**, a value specified on a line in the textual input, or a fallback value of 1. If the addition causes an overflow of the maximum counter value (18446744073709551614), the counter is set to the maximum. A message is printed to the standard error the first time an overflow condition is detected.

SET INPUT

When creating a Bag from an IPset, the count associated with each IP address is the value specified by the **--default-count** switch or 1 if the switch is not provided.

If the **--key-type** is `sip-country`, `dip-country`, or `any-country`, each IP address is mapped to its country code using the country code mapping file (see FILES) and that key is added to the Bag file with the **--default-count** value.

If the **--key-type** is `sip-pmap`, `dip-pmap`, or `any-ip-pmap`, each IP address is mapped to a value found in the prefix map file specified in **--pmap-file** and that value is added to the Bag file with the **--default-count** value.

BAG (TEXTUAL) INPUT

The textual input read from the argument to the **--bag-input** switch is processed a line at a time. Comments begin with a '#'-character and continue to the end of the line; they are stripped from each line. Any line that is blank or contains only whitespace is ignored. All other lines must contain a valid key or key-counter pair; whitespace around the key and counter is ignored. The key and counter are separated by a one-character delimiter. The default delimiter is vertical bar ('|'); use **--delimiter** to specify a different delimiter.

Each line that is not ignored must begin with a key. The accepted formats of the key are described below.

When the **--default-count** switch is given, **rwbagtool** only parses the key and ignores everything on a line to the right of the first delimiter. To re-iterate, the **--default-count** switch overrides any counter present on the line.

If the delimiter is not present on a line, **rwbagtool** parses the key and adds the **--default-count** value (or the fallback value of 1) to the Bag for that key.

When **--default-count** is not given, any text between the first delimiter and optional second delimiter on a line is treated as the counter. If the counter contains only whitespace, the counter for the key is incremented by 1; otherwise, the counter must be a (decimal) number from 0 to 18446744073709551614 inclusive. If a second delimiter is present, it and any text that follows it is ignored.

rwbagbuild prints an error and exits when a key or counter cannot be parsed.

Format of the counter

The counter is any non-negative (decimal) integer value from 0 to 18446744073709551614 inclusive (the maximum is one less than the maximum unsigned 64-bit value). When writing the Bag file, keys whose counter is zero are not written to the file.

Format of the Key

The key is a 32-bit integer, an IP address, a CIDR block, a SiLK IPWildcard, or a pair of numbers when the key-type is a protocol-port prefix map file.

For key-types that use fewer than 32-bits, **rwbagbuild** does *not* verify the validity of the key. For example, it is possible to have 257 as a key in Bag whose key-type is protocol.

rwbagbuild parses specific key-types as follows:

sIPv4, dIPv4, nhIPv4, any-IPv4

key is an IPv4 address or a 32-bit value; key-type set to corresponding IPv6 type when an IPv6 address is present. A CIDR block or SiLK IPWildcard representing multiple addresses adds multiple entries to the Bag

sIPv6, dIPv6, nhIPv6, any-IPv6

key is an IPv6 address. An IPv4 address is mapped into the ::ffff:0:0/96 netblock. All keys must be IP addresses (integers are not allowed).

flags, initialFlags, sessionFlags

key is the numeric value of the flags, 17 = FIN|ACK

sTime, eTime, any-time

key is seconds since the UNIX epoch

duration

key represents seconds

sensor

key is the numeric sensor ID

sip-country, dip-country, any-country

key is an IP address; the *country_codes.pmap* prefix map file is used to map the IP to a country code that is stored in the Bag

sip-pmap, dip-pmap, any-ip-pmap

key is an IP address; the specified **--prefix-map** file is used to map the IP to a value that is stored in the Bag

sport-pmap, dport-pmap, any-port-pmap

key is comprised of two numbers separated by a delimiter: a protocol (8-bit number) and a port (16-bit number). Those values are looked up in the specified **--prefix-map** file and the result is stored in the Bag. The delimiter separating the protocol and port may be set by **--proto-port-delimiter**. If not explicitly set, it is the same as the delimiter specified to **--delimiter**. The default delimiter is '|'.

attributes

these bits of the key are relevant, though any 32-bit value is accepted: 0x08=F, 0x10=S, 0x20=T, 0x40=C

class, type

key is treated as a number

An IP address or integer key must be expressed in one of the following formats. **rwbagbuild** complains if the key field contains a mixture of IPv6 addresses and integer values.

- Dotted decimal---all 4 octets are required:

```
10.1.2.4
```

- An unsigned 32-bit integer:

```
167838212
```

- An IPv6 address in canonical format (when SiLK has been compiled with IPv6 support):

```
2001:db8:a:1::2:4
::ffff:10.1.2.4
```

- Any of the above with a CIDR designation---for dotted decimal all four octets are still required:

```
10.1.2.4/31
167838212/31
2001:db8:a:1::2:4/127
::ffff:10.1.2.4/31
```

- SiLK IP wildcard notation. A SiLK IP Wildcard can represent multiple IPv4 or IPv6 addresses. An IP Wildcard contains an IP in its canonical format, except each part of the IP (where *part* is an octet for IPv4 or a hexadectet for IPv6) may be a single value, a range, a comma separated list of values and ranges, or the letter `x` to signify all values for that part of the IP (that is, `0-255` for IPv4). You may not specify a CIDR suffix when using the IP Wildcard notation.

```
10.x.1-2.4,5
2001:db8:a:x::1-2:4,5
```

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

The first two switches control the type of input; exactly one must be provided:

--set-input=*SETFILE*

Create a Bag from an IPset. *SETFILE* is a filename, a named pipe, or the keyword `stdin` or `-` to read the IPset from the standard input. Counts have a volume of 1 when the **--default-count** switch is not specified. (IPsets are typically created by **rwset(1)** or **rwsetbuild(1)**.)

--bag-input=*TEXTFILE*

Create a Bag from a delimited text file. *TEXTFILE* is a filename, a named pipe, or the keyword `stdin` or `-` to read the text from the standard input. See the DESCRIPTION section for the syntax of the *TEXTFILE*.

--delimiter=*C*

Expect the character *C* between each key-counter pair in the *TEXTFILE* read by the **--bag-input** switch. The default delimiter is the vertical pipe ('|'). The delimiter is ignored if the **--set-input** switch is specified. When the delimiter is a whitespace character, any amount of whitespace may surround and separate the key and counter. Since '`#`' is used to denote comments and newline is used to denote records, neither is a valid delimiter character.

--proto-port-delimiter=*C*

Expect the character *C* between the protocol and port that comprise a key when the **--key-type** is `sport-pmap`, `dport-pmap`, or `any-port-pmap`. Unless this switch is specified, **rwbagbuild** expects the key-counter delimiter to appear between the protocol and port.

--default-count=*DEFAULTCOUNT*

Override the counts of **all** values in the input text or IPset with the value of *DEFAULTCOUNT*. *DEFAULTCOUNT* must be a positive integer from 1 to 18446744073709551614 inclusive.

--key-type=*FIELD_TYPE*

Write a entry into the header of the Bag file that specifies the key contains *FIELD_TYPE* values. When this switch is not specified, the key type of the Bag is set to `custom`. The *FIELD_TYPE* is case insensitive. The supported *FIELD_TYPE*s are:

sIPv4
: source IP address, IPv4 only

dIPv4
: destination IP address, IPv4 only

sPort
: source port

dPort
: destination port

protocol
: IP protocol

packets
: packets, see also `sum-packets`

bytes
: bytes, see also `sum-bytes`

flags
: an unsigned bitwise OR of TCP flags

sTime
: starting time of the flow record, seconds resolution

duration
: duration of the flow record, seconds resolution

eTime
: ending time of the flow record, seconds resolution

sensor
: sensor ID

input
: SNMP input

output
: SNMP output

nhIPv4
: next hop IP address, IPv4 only

initialFlags
: TCP flags on first packet in the flow

sessionFlags
: bitwise OR of TCP flags on all packets in the flow except the first

attributes
: flow attributes set by the flow generator

application
: guess as to the content of the flow, as set by the flow generator

class
: class of the sensor

type
: type of the sensor

icmpTypeCode
: an encoded version of the ICMP type and code, where the type is in the upper byte and the code is in the lower byte

sIPv6
: source IP, IPv6

dIPv6
: destination IP, IPv6

nhIPv6
: next hop IP, IPv6

records
: count of flows

sum-packets
sum of packet counts

sum-bytes
sum of byte counts

sum-duration
sum of duration values

any-IPv4
a generic IPv4 address

any-IPv6
a generic IPv6 address

any-port
a generic port

any-snmp
a generic SNMP value

any-time
a generic time value, in seconds resolution

sip-country
the country code of the source IP address. For textual input, the key column must contain an IP address or an integer. **rwbagbuild** maps the IP address to a country code and stores the country code in the bag. Uses the mapping file specified by the SILK_COUNTRY_CODES environment variable or the *country_codes.pmap* mapping file, as described in FILES. (See also **ccfilter(3)**.) *Since SiLK 3.12.0.*

dip-country
the country code of the destination IP. See `sip-country`. *Since SiLK 3.12.0.*

any-country
the country code of any IP address. See `sip-country`. *Since SiLK 3.12.0.*

sip-pmap
a prefix map value found from a source IP address. Maps each IP address in the key column to a value from a prefix map file and stores the value in the bag. The type of the prefix map must be IPv4-address or IPv4-address. Use the **--pmap-file** switch to specify the path to the file. *Since SiLK 3.12.0.*

dip-pmap
a prefix map value found from a destination IP address. See `sip-pmap`. *Since SiLK 3.12.0.*

any-ip-pmap:*PMAP_PATH*
a prefix map value found from any IP address. See `sip-pmap`. *Since SiLK 3.12.0.*

sport-pmap
a prefix map value found from a protocol/source-port pair. Each key must contain two values, a protocol and a port. Maps each protocol/port pair to a value from a prefix map file and stores the value in the bag. The type of the prefix map must be proto-port. Use the **--pmap-file** switch to specify the path to the file. *Since SiLK 3.12.0.*

dport-pmap
a prefix map value found from a protocol/destination-port pair. See `sport-pmap`. *Since SiLK 3.12.0.*

any-port-pmap
a prefix map value found from a protocol/port pair. See `sport-pmap`. *Since SiLK 3.12.0.*

custom
: a number

--counter-type=*FIELD_TYPE*

Write a entry into the header of the Bag file that specifies the counter contains *FIELD_TYPE* values. When this switch is not specified, the counter type of the Bag is set to `custom`. Although the supported *FIELD_TYPE*s are the same as those for the key, the value is always treated as a number that can be summed. **rwbagbuild** does not use the country code or prefix map when parsing the value field.

--pmap-file=*PATH*

--pmap-file=*MAPNAME*:*PATH*

When the key-type is one of `sip-pmap`, `dip-pmap`, `any-ip-pmap`, `sport-pmap`, `dport-pmap`, or `any-port-pmap`, use the prefix map file located at *PATH* to map the key to a string. Specify *PATH* as `-` or `stdin` to read from the standard input. A map-name may be included in the argument to the switch, but **rwbagbuild** currently does not use the map-name. To create a prefix map file, use **rwpmapbuild(1)**. *Since SiLK 3.12.0.*

--note-add=*TEXT*

Add the specified *TEXT* to the header of the output file as an annotation. This switch may be repeated to add multiple annotations to a file. To view the annotations, use the **rwfileinfo(1)** tool.

--note-file-add=*FILENAME*

Open *FILENAME* and add the contents of that file to the header of the output file as an annotation. This switch may be repeated to add multiple annotations. Currently the application makes no effort to ensure that *FILENAME* contains text; be careful that you do not attempt to add a SiLK data file as an annotation.

--invocation-strip

Do not record the command used to create the Bag file in the output. When this switch is not given, the invocation is written to the file's header, and the invocation may be viewed with **rwfileinfo(1)**. *Since SiLK 3.12.0.*

--compression-method=*COMP_METHOD*

Specify the compression library to use when writing output files. If this switch is not given, the value in the SILK_COMPRESSION_METHOD environment variable is used if the value names an available compression method. When no compression method is specified, output to the standard output or to named pipes is not compressed, and output to files is compressed using the default chosen when SiLK was compiled. The valid values for *COMP_METHOD* are determined by which external libraries were found when SiLK was compiled. To see the available compression methods and the default method, use the **--help** or **--version** switch. SiLK can support the following *COMP_METHOD* values when the required libraries are available.

none
: Do not compress the output using an external library.

zlib
: Use the **zlib(3)** library for compressing the output, and always compress the output regardless of the destination. Using zlib produces the smallest output files at the cost of speed.

lzo1x
: Use the *lzo1x* algorithm from the LZO real time compression library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead.

snappy
: Use the *snappy* library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead. *Since SiLK 3.13.0.*

best
: Use lzo1x if available, otherwise use snappy if available, otherwise use zlib if available. Only compress the output when writing to a file.

--output-path=*PATH*
: Write the binary Bag output to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output. If *PATH* names an existing file, **rwbagtool** exits with an error unless the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. If this switch is not given, the output is written to the standard output. Attempting to write the binary output to a terminal causes **rwbagtool** to exit with an error.

--help
: Print the available options and exit.

--version
: Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

Create a bag with IP addresses as keys from a text file

Assume the file *mybag.txt* contains the following lines, where each line contains an IP address, a comma as a delimiter, a count, and ends with a newline.

```
192.168.0.1,5
192.168.0.2,500
192.168.0.3,3
192.168.0.4,14
192.168.0.5,5
```

To build a bag with it:

```
$ rwbagbuild --bag-input=mybag.txt --delimiter=, > mybag.bag
```

Use **rwbagcat(1)** to view its contents:

```
$ rwbagcat mybag.bag
    192.168.0.1|                  5|
    192.168.0.2|                500|
    192.168.0.3|                  3|
    192.168.0.4|                 14|
    192.168.0.5|                  5|
```

Create a bag with protocols as keys from a text file

To create a Bag of protocol data from the text file *myproto.txt*:

```
  1|      4|
  6|    138|
 17|    131|
```

use

```
$ rwbagbuild --key-type=proto --bag-input=myproto.txt > myproto.bag
$ rwbagcat myproto.bag
         1|                   4|
         6|                 138|
        17|                 131|
```

When the **--key-type** switch is specified, **rwbagcat** knows the keys should be printed as integers, and **rwfileinfo(1)** shows the type of the key:

```
$ rwfileinfo --fields=bag myproto.bag
myproto.bag:
  bag            key: protocol @ 4 octets; counter: custom @ 8 octets
```

Without the **--key-type** switch, **rwbagbuild** assumes the integers in *myproto.txt* represent IP addresses:

```
$ rwbagbuild --bag-input=myproto.txt | rwbagcat
        0.0.0.1|                   4|
        0.0.0.6|                 138|
       0.0.0.17|                 131|
```

Although the **--key-format** switch on **rwbagcat** may be used to choose how the keys are displayed, it is generally better to use the **--key-type** switch when creating the bag.

$ rwbagbuild --bag-input=myproto.txt | rwbagcat --key-format=decimal 1| 4| 6| 138| 17| 131|

Create a bag and override the existing counter

To ignore the counts that exist in *myproto.txt* and set the counts for each protocol to 1, use the **--default-count** switch which overrides the existing value:

```
$ rwbagbuild --key-type=protocol --bag-input=myproto.txt  \
      --default-count=1 --output-path=myproto1.bag
$ rwbagcat myproto1.bag
         1|                   1|
         6|                   1|
        17|                   1|
```

Create a bag from multiple text files

To create a bag from multiple text files (*X.txt*, *Y.txt*, and *Z.txt*), use the UNIX **cat(1)** utility to concatenate the files and have **rwbagbuild** read the combined input. To avoid creating a temporary file, feed the output of **cat** as the standard input to **rwbagbuild**.

```
$ cat X.txt Y.txt Z.txt                                  \
  | rwbagbuild --bag-input=- --output-path=xyz.bag
```

For each key that appears in multiple input files, **rwbagbuild** sums the counters for the key.

Create a bag with IP addresses as keys from an IPset file

Given the IP set *myset.set*, create a bag where every entry in the bag has a count of 3:

```
$ rwbagbuild --set-input=myset.set --default-count=3  \
       --out=mybag2.bag
```

Create a bag from multiple IPset files

Suppose we have three IPset files, *A.set*, *B.set*, and *C.set*:

```
$ rwsetcat A.set
10.0.0.1
10.0.0.2
$ rwsetcat B.set
10.0.0.2
10.0.0.3
$ rwsetcat C.set
10.0.0.1
10.0.0.2
10.0.0.4
```

We want to create a bag file from these IPset files where the count for each IP address is the number of files that IP appears in. **rwbagbuild** accepts a single file as an argument, so we cannot do the following:

```
$ rwbagbuild --set-input=A.set --set-input=B.set ...   # WRONG!
```

(Even if we could repeat the **--set-input** switch, specifying it multiple times would be annoying if we had 300 files instead of only 3.)

Since IPset files are (mathematical) sets, joining them together first with **rwsettool(1)** and then running **rwbagbuild** causes each IP address to get a count of 1:

```
$ rwsettool --union A.set B.set C.set   \
  | rwbagbuild --set-input=-            \
  | rwbagcat
       10.0.0.1|                  1|
       10.0.0.2|                  1|
       10.0.0.3|                  1|
       10.0.0.4|                  1|
```

When **rwbagbuild** is processing textual input, it sums the counters for keys that appear in the input multiple times. We can use **rwsetcat(1)** to convert each IPset file to text and feed that as single textual stream to **rwbagbuild**. Use the **--cidr-blocks** switch on **rwsetcat** to reduce the amount of input that **rwbagbuild** must process. This is probably the best approach to the problem:

```
$ rwsetcat --cidr-block *.set | rwbagbuild --bag-input=- > total1.bag
$ rwbagcat total1.bag
       10.0.0.1|                   2|
       10.0.0.2|                   3|
       10.0.0.3|                   1|
       10.0.0.4|                   1|
```

A less efficient solution is to convert each IPset to a bag and then use **rwbagtool(1)** to add the bags together:

```
$ for i in *.set ; do
       rwbagbuild --set-input=$i --output-path=/tmp/$i.bag ;
  done
$ rwbagtool --add /tmp/*.set.bag > total2.bag
$ rm /tmp/*.set.bag
```

There is no need to create a bag file for each IPset; we can get by with only two bag files, the final bag file, *total3.bag*, and a temporary file, *tmp.bag*. We initialize *total3.bag* to an empty bag. As we loop over each IPset, **rwbagbuild** converts the IPset to a bag on its standard output, **rwbagtool** creates *tmp.bag* by adding its standard input to *total3.bag*, and we rename *tmp.bag* to *total3.bag*:

```
$ rwbagbuild --bag-input=/dev/null --output-path=total3.bag
$ for i in *.set ; do
       rwbagbuild --set-input=$i  \
       | rwbagtool --output-path=tmp.bag --add total3.bag stdin ;
       /bin/mv tmp.bag total3.bag ;
  done
$ rwbagcat total3.bag
       10.0.0.1|                   2|
       10.0.0.2|                   3|
       10.0.0.3|                   1|
       10.0.0.4|                   1|
```

Create a bag where the key is the country code

As of SiLK 3.12.0, a Bag file may contain a country code as its key. In **rwbagbuild**, specify the **--key-type** as `sip-country`, `dip-country`, or `any-country`. That key-type works with either textual input or IPset input. The form of the textual input when mapping an IP address to a country code is identical to that when building an ordinary bag.

```
$ rwbagbuild --bag-input=mybag.txt --delimiter=,       \
       --key-type=any-country --output-path=scc1.bag
$ rwbagcat scc1.bag
--|                 527|
```

```
$ rwbagbuild --set-input=A.set --key-type=any-country  \
       --output-path=scc2.bag
$ rwbagcat scc2.bag
--|                  2|
```

Create a bag using a prefix map value as the key

rwbagbuild and **rwbag(1)** can use a prefix map file as the key in a Bag file as of SiLK 3.12.0. Use the **--pmap-file** switch to specify the prefix map file, and specify the **--key-type** using one of the types that end in `-pmap`.

For a prefix map that maps by IP addresses, use a key-type of `sip-pmap`, `dip-pmap`, or `any-ip-pmap`. The input may be an IPset or text. The form of the textual input is the same as for a normal bag file.

```
$ rwbagbuild --set-input=A.set --key-type=sip-pmap     \
       --pmap-file=ip-map.pmap --output=test1.bag
```

```
$ rwbagbuild --bag-input=mybag.txt --delimiter=,       \
       --key-type=sip-pmap --pmap-file=ip-map.pmap     \
       --output-path=test2.bag
```

The prefix map file is not stored as part of the Bag, so you must provide the name of the prefix map when running **rwbagcat(1)**.

```
$ rwbagcat --pmap-file=ip-map.pmap test2.bag
         internal|                 527|
```

For a prefix map file that maps by protocol-port pairs, the textual input must contain either three column (protocol, port, counter) or two columns (protocol and port) which uses the **--default-counter**.

```
$ cat proto-port-count.txt
6| 25|  800|
6| 80| 5642|
6| 22
$ rwbagbuild --key-type=sport-pmap                \
       --bag-input=proto-port-count.txt           \
       --pmap-file=proto-port-map.pmap            \
       --output-path=service.bag
$ rwbagcat --pmap-file=port-map.pmap service.bag
  TCP/SSH|                   1|
 TCP/SMTP|                 800|
 TCP/HTTP|                5642|
```

Delimiter examples

A single value followed by an optional delimiter is treated as a key. The counter for those keys is set to 1. A delimiter may follow the count, and any text after that delimiter is ignored. When the counter is 0, the key is not inserted into the Bag.

```
$ cat sport.txt
0
1|
2|3
4|5|
6|7|8|
9|10|||||
11|0
$ rwbagbuild --bag-input=sport.txt --key-type=sport \
  | rwbagcat
         0|                   1|
         1|                   1|
         2|                   3|
         4|                   5|
         6|                   7|
         9|                  10|
```

The **--default-counter** switch overrides the count.

```
$ rwbagbuild --bag-input=sport.txt --key-type=sport --default-count=1 \
  | rwbagcat
         0|                   1|
         1|                   1|
         2|                   1|
         4|                   1|
         6|                   1|
         9|                   1|
        11|                   1|
```

In fact, the **--default-counter** switch causes **rwbagbuild** to ignore all text after the delimiter that follows the key.

```
$ echo '12|13 14' | rwbagbuild --bag-input=- --output=/dev/null
rwbagbuild: Error parsing line 1: Extra text after count
rwbagbuild: Error creating bag from text bag

$ echo '12|13 14' | rwbagbuild --bag-input=- --default-count=1 \
  | rwbagcat --key-format=decimal
        12|                   1|
```

ENVIRONMENT

SILK_COUNTRY_CODES

This environment variable allows the user to specify the country code mapping file that **rwbagbuild** uses when mapping an IP to a country for the `sip-country`, `dip-country`, or `any-country` keys. The value may be a complete path or a file relative to the SILK_PATH. See the FILES section for standard locations of this file.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_COMPRESSION_METHOD

This environment variable is used as the value for **--compression-method** when that switch is not provided. *Since SiLK 3.13.0.*

SILK_PATH

This environment variable gives the root of the install tree. When searching for the country code mapping file, **rwbagbuild** may use this environment variable. See the FILES section for details.

FILES

$SILK_COUNTRY_CODES

$SILK_PATH/share/silk/country_codes.pmap

$SILK_PATH/share/country_codes.pmap

/usr/local/share/silk/country_codes.pmap

/usr/local/share/country_codes.pmap

Possible locations for the country code mapping file required by the `sip-country`, `dip-country`, and `any-country` key-types.

SEE ALSO

rwbag(1), **rwbagcat(1)**, **rwbagtool(1)**, **rwfileinfo(1)**, **rwpmapbuild(1)**, **rwset(1)**, **rwsetbuild(1)**, **rwsetcat(1)**, **rwsettool(1)**, **ccfilter(3)**, **silk(7)**, **zlib(3)**, **cat(1)**

BUGS

rwbagbuild should verify the key's value is within the allowed range for the specified **--key-type**.

rwbagbuild should accept non-numeric values for some fields, such as times and TCP flags.

The **--default-count** switch is poorly named.

rwbagcat

Output a binary Bag file as text

SYNOPSIS

```
rwbagcat [ --network-structure[=STRUCTURE] | --bin-ips[=SCALE]
           | --sort-counters[=ORDER]]
      [--print-statistics[=OUTFILE]]
      [--minkey=VALUE] [--maxkey=VALUE] [--mask-set=PATH]
      [--mincounter=VALUE] [--maxcounter=VALUE] [--zero-counts]
      [{ --pmap-file=PATH | --pmap-file=MAPNAME:PATH }]
      [--key-format=FORMAT] [--integer-keys] [--zero-pad-ips]
      [--no-columns] [--column-separator=C]
      [--no-final-delimiter] [{--delimited | --delimited=C}]
      [--output-path=PATH] [--pager=PAGER_PROG]
      [--site-config-file=FILENAME]
      [BAGFILE [BAGFILE...]]

rwbagcat --help

rwbagcat --version
```

DESCRIPTION

rwbagcat reads a binary Bag as created by **rwbag(1)** or **rwbagbuild(1)**, converts it to text, and writes it to the standard output, to the pager, or to the specified output file. It can also print various statistics and summary information about the Bag.

As of SiLK 3.12.0, **rwbagcat** uses information in the Bag file's header to determine how to display the key column.

- A key that is an IP address is printed in the canonical format. Specifically, IPs are printed in the IPv4 canonical format if the Bag contains only IPv4 addresses; otherwise, in the IPv6 canonical format (with IPv4 mapped into the ::ffff:0:0/96 netblock). May be modified by **--key-format**.
- A key that is a time is printed as a human-readable timestamp. May be modified by **--key-format**.
- A sensor key prints the name of the sensor. The `decimal` and `hexadecimal` arguments to **--key-format** may be used.
- A key holding TCP Flags is printed using the characters `F,S,R,P,A,U,E,C`. The `decimal` and `hexadecimal` arguments to **--key-format** may be used.
- A key holding SiLK attributes is printed using the characters `T,C,F,S`. The `decimal` and `hexadecimal` arguments to **--key-format** may be used.
- A country code key uses the abbreviations defined by ISO 3166-1 (see for example https://www.iso.org/iso-3166-country-codes.html or https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2) or the following special codes: **--** N/A (e.g. private and experimental reserved addresses); **a1** anonymous proxy; **a2** satellite provider; **o1** other.

- A key holding a value from prefix map requires that the **--pmap-file** switch be specified to display the value.

In addition, **rwbagcat** exits with an error when asked to use an IP format to display keys that are not IP addresses.

rwbagcat reads the *BAGFILE*s specified on the command line; if no *BAGFILE* arguments are given, **rwbagcat** attempts to read the Bag from the standard input. *BAGFILE* may be the keyword `stdin` or a hyphen (`-`) to allow **rwbagcat** to print data from both files and piped input. If any input does not contain a Bag, **rwbagcat** prints an error to the standard error and exits abnormally.

When multiple *BAGFILE*s are specified on the command line, each is handled individually. To process the files as a single Bag, use **rwbagtool(1)** to combine the bags and pipe the output of **rwbagtool** into **rwbagcat**.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--network-structure

--network-structure=*STRUCTURE*

For each numeric value in *STRUCTURE*, group the IPs in the Bag into a netblock of that size and print the number of hosts, the sum of the counters, and, optionally, print the number of smaller, occupied netblocks that each larger netblock contains. When *STRUCTURE* begins with `v6:`, the IPs in the Bag are treated as IPv6 addresses, and any IPv4 addresses are mapped into the ::ffff:0:0/96 netblock. Otherwise, the IPs are treated as IPv4 addresses, and any IPv6 address outside the ::ffff:0:0/96 netblock is ignored. Aside from the initial `v6:` (or `v4:`, for consistency), *STRUCTURE* has one of following forms:

1. *NETBLOCK_LIST*/*SUMMARY_LIST*. Group IPs into the sizes specified in either *NETBLOCK_LIST* or *SUMMARY_LIST*. **rwbagcat** prints a row for each occupied netblock specified in *NETBLOCK_LIST*, where the row lists the base IP of the netblock, the sum of the counters for that netblock, the number of hosts, and the number of smaller, occupied netblocks having a size that appears in either *NETBLOCK_LIST* or *SUMMARY_LIST*. (The values in *SUMMARY_LIST* are only summarized; they are not printed.)
2. *NETBLOCK_LIST*`/`. Similar to the first form, except all occupied netblocks are printed, and there are no netblocks that are only summarized.
3. *NETBLOCK_LIST***S**. When the character `S` appears anywhere in the *NETBLOCK_LIST*, **rwbagcat** provides a default value for the *SUMMARY_LIST*. That default is 8,16,24,27 for IPv4, and 48,64 for IPv6.
4. *NETBLOCK_LIST*. When neither `S` nor `/` appear in *STRUCTURE*, the output does not include the number of smaller, occupied netblocks.
5. Empty. When *STRUCTURE* is empty or only contains `v6:` or `v4:`, the *NETBLOCK_LIST* prints a single row for the total network (the /0 netblock) giving the number of hosts, the sum of the counters, and the number of smaller, occupied netblocks using the same default list specified in form 3.

NETBLOCK_LIST and *SUMMARY_LIST* contain a comma separated list of numbers between 0 (the total network) and the size for an individual host (32 for IPv4 or 128 for IPv6). The characters `T` and `H` may be used as aliases for 0 and the host netblock, respectively. In addition, when parsing the lists as IPv4 netblocks, the characters `A`, `B`, `C`, and `X` are supported as aliases for 8, 16, 24, and 27, respectively. A comma is not required between adjacent letters. The **--network-structure** switch disables printing of the IPs in the Bag file; specify the `H` argument to the switch to print each individual IP address and its counter.

The **--network-structure** switch may not be combined with the **--bin-ips** or **--sort-counters** switches. As of SiLK 3.12.0, **rwbagcat** exits with an error if the **--network-structure** switch is used on a Bag file whose key-type is neither `custom` nor an IP address type.

--bin-ips

--bin-ips=*SCALE*

Invert the bag and count the total number of unique keys for a given value of the volume bin. For example, turn a Bag {sip:flow} into {flow:count(sip)}. *SCALE* is a string containing the value `linear`, `binary`, or `decimal`.

- The default behavior is `linear`: Each distinct counter gets its own bin. Any counter in the input Bag file that is larger than the maximum possible key will be attributed to the maximum key; to prevent this, specify `--maxcounter=4294967295` which discards bins whose counter value does not fit into a key.
- `binary` creates a bag of {log2(flow):count(sip)}. Bin `n` contains counts in the range `[ 2^n, 2^(n+1) )`.
- `decimal` creates one hundred bins for each counter in the range [1,100), and one hundred bins for each counter in the range [100,1000), each counter in the range [1000,10000), etc. Counters are logarithmically distributed among the bins.

The **--bin-ips** switch may not be combined with the **--network-structure** or **--sort-counters** switches. See also the **--invert** switch on **rwbagtool(1)** which inverts a bag using a linear scale and creates a new binary bag file.

--sort-counters

--sort-counters=*ORDER*

Sort the output so the counters are presented in either decreasing or increasing order. Typically the output is sorted by the keys. If the *ORDER* argument is not given to the switch, the counters are printed in decreasing order. Valid values for *ORDER* are

decreasing
: Print the maximum counter first. This is the default.

increasing
: Print the minimum counter first.

When two counters have the same value, the smaller key is displayed first. The **--sort-counters** switch may not be combined with the **--network-structure** or **--bin-ips** switches. *Since SiLK 3.12.2.*

--print-statistics

--print-statistics=*OUTFILE*

Print a breakdown of the network hosts seen, and print general statistics about the keys and counters. When **--print-statistics** is specified, no other output is produced unless one of **--sort-counters**, **--network-structure**, or **--bin-ips** is also specified. When the *OUTFILE* argument is not given, the

statistics are written to the standard output or to the pager if output is to a terminal. *OUTFILE* is a filename, named pipe, the keyword `stderr` to write to the standard error, or the keyword `stdout` or `-` to write to the standard output. If *OUTFILE* names an existing file, **rwbagcat** exits with an error unless the SILK_CLOBBER environment variable is set, in which case *OUTFILE* is overwritten. The output statistics produced by this switch are:

- count of unique keys
- sum of all the counters
- minimum key
- maximum key
- minimum counter
- maximum counter
- mean of counters
- variance of counters
- standard deviation of counters
- skew of counters
- kurtosis of counters
- count of nodes allocated
- total bytes allocated for nodes
- count of leaves allocated
- total bytes allocated for leaves
- density of the data

--minkey=*VALUE*

Output records whose key value is at least *VALUE*. *VALUE* may be an IP address or an integer in the range 0 to 4294967295 inclusive. The default is to print all records with a non-zero counter.

--maxkey=*VALUE*

Output records whose key value is not more than *VALUE*. *VALUE* may be an IP address or an integer in the range 0 to 4294967295 inclusive. The default is to print all records with a non-zero counter.

--mask-set=*PATH*

Output records whose key appears in the binary IPset read from the file *PATH*. (To build an IPset, use **rwset(1)** or **rwsetbuild(1)**.) When used with **--minkey** and/or **--maxkey**, output records whose key is in the IPset and is also within when the specified range. As of SiLK 3.12.0, **rwbagcat** exits with an error if the **--mask-set** switch is used on a Bag file whose key-type is neither `custom` nor an IP address type.

--mincounter=*VALUE*

Output records whose counter value is at least *VALUE*. *VALUE* is an integer in the range 1 to 18446744073709551615. The default is to print all records with a non-zero counter; use **--zero-counts** to show records whose counter is 0.

--maxcounter=*VALUE*

Output records whose counter value is not more than *VALUE*. *VALUE* is an integer in the range 1 to 18446744073709551615, with the default being the maximum counter value.

--zero-counts

Print keys whose counter is zero. Normally, keys with a counter of zero are suppressed since all keys have a default counter of zero. In order to use this flag, either **--mask-set** or both **--minkey** and **--maxkey** must be specified. When this switch is specified, any counter limit explicitly set by the **--maxcounter** switch is also applied.

--pmap-file=*PATH*

--pmap-file=*MAPNAME*:*PATH*

Use the prefix map file located at *PATH* to map the key to a string when the type of the Bag's key is one of `sip-pmap`, `dip-pmap`, `any-ip-pmap`, `sport-pmap`, `dport-pmap`, or `any-port-pmap`. This switch is required for Bag files whose key was derived from a prefix map file. The type of the prefix map file must match the key's type, but a different prefix map file may be used. Specify *PATH* as `-` or `stdin` to read from the standard input. A map-name may be included in the argument to the switch, but **rwbagcat** currently does not use the map-name. To create a prefix map file, use **rwpmapbuild(1)**. *Since SiLK 3.12.0.*

--key-format=*FORMAT*

Specify the format to use when printing a key, where *FORMAT* is a comma-separated list of the arguments described below. When this switch is not specified, **rwbagcat** uses the key's type to determine how to format the key, and a key whose type is unknown or `custom` is assumed to be an IP address. **rwbagcat** exits with an error if the specified format is incompatible with the key's type (for example, attempting to format a timestamp as an IP address).

decimal

Print keys as integers in decimal format. For example, print `192.0.2.1` and `2001:db8::1` as `3221225985` and `42540766411282592856903984951653826561`, respectively. May be combined with `zero-padded` and either `map-v4` or `unmap-v6`. **rwbagcat** exits with an error when this format is used on a Bag file whose key-type is a timestamp.

hexadecimal

Print keys as integers in hexadecimal format. For example, print `192.0.2.1` and `2001:db8::1` as `c00000201` and `20010db800000000000000000000001`, respectively. May be combined with `zero-padded` and either `map-v4` or `unmap-v6`. **rwbagcat** exits with an error when this format is used on a Bag file whose key-type is a timestamp. **Note:** This setting does not apply to CIDR prefix values which are printed as decimal.

canonical

Print keys as IP addresses in the canonical format. If the key is an IPv4 address, use dotted decimal (`192.0.2.1`). If the key is an IPv6 address, use colon-separated hexadecimal (`2001:db8::1`) or a mixed IPv4-IPv6 representation for IPv4-mapped IPv6 addresses (the ::ffff:0:0/96 netblock, e.g., `::ffff:192.0.2.1`) and IPv4-compatible IPv6 addresses (the ::/96 netblock other than ::/127, e.g., `::192.0.2.1`). May be combined with `zero-padded` and either `map-v4` or `unmap-v6`. As of SiLK 3.12.0, **rwbagcat** exits with an error when this format is used on a Bag file whose key-type is neither `custom` nor an IP address type.

no-mixed

Print keys as IP addresses in the canonical format (`192.0.2.1` or `2001:db8::1`) but do not used the mixed IPv4-IPv6 representations. For example, use `::ffff:c000:201` instead of `::ffff:192.0.2.1`. May be combined with `zero-padded` and either `map-v4` or `unmap-v6`. **rwbagcat** exits with an error when this format is used on a Bag file whose key-type is neither `custom` nor an IP address type. *Since SiLK 3.17.0.*

map-v4

When the Bag's key is an IPv4 address, change all IPv4 addresses to IPv4-mapped IPv6 addresses (addresses in the ::ffff:0:0/96 netblock) prior to formatting. May be combined with one of the above settings. **rwbagcat** exits with an error when this format is used on a Bag file whose key-type is neither `custom` nor an IP address type. *Since SiLK 3.17.0.*

unmap-v6

When the Bag's key is an IPv6 address, change any IPv4-mapped IPv6 addresses (addresses in the ::ffff:0:0/96 netblock) to IPv4 addresses prior to formatting. May be combined with any one of the above settings except `map-v4`. **rwbagcat** exits with an error when this format is used on a Bag file whose key-type is neither `custom` nor an IP address type. *Since SiLK 3.17.0.*

zero-padded

Make all formatted key strings contain the same number of characters by padding numbers with leading zeros. For example, print `192.0.2.1` and `2001:db8::1` as `192.000.002.001` and `2001:0db8:0000:0000:0000:0000:0000:0001`, respectively. For IPv6 addresses, this setting implies `no-mixed`, so that `::ffff:192.0.2.1` is printed as `0000:0000:0000:0000:0000:ffff:c000:0201`. As of SiLK 3.17.0, may be combined with any of the above, including `decimal` and `hexadecimal`. As of SiLK 3.18.0, the values of CIDR prefix are also zero-padded. **rwbagcat** exits with an error when this format is used on a Bag file whose key-type is a timestamp.

force-ipv6

Print keys using the format `map-v4`,`no-mixed`. May be combined with `zero-padded`. As of SiLK 3.12.0, **rwbagcat** exits with an error when this format is used on a Bag file whose key-type is neither `custom` nor an IP address type.

timestamp

Print keys as time in standard SiLK format: *yyyy*/*mm*/*dd*T*hh*:*mm*:*ss*. May be combined with **utc** or **localtime**. May only be used on keys whose type is `custom` or a time value. *Since SiLK 3.12.0.*

iso-time

Print keys as time in the ISO time format *yyyy-mm-dd hh*:*mm*:*ss*. May be combined with **utc** or **localtime**. May only be used on keys whose type is `custom` or a time value. *Since SiLK 3.12.0.*

m/d/y

Print keys as time in the format *mm*/*dd*/*yyyy hh*:*mm*:*ss*. May be combined with **utc** or **localtime**. May only be used on keys whose type is `custom` or a time value. *Since SiLK 3.12.0.*

utc

Print the keys as time in UTC. If no other time-related key-format is provided, formats the time using the **timestamp** format. May only be used on keys whose type is `custom` or a time value. *Since SiLK 3.12.0.*

localtime

Print as the keys as time and get the timezone from either the TZ environment variable or local machine. If no other time-related key-format is provided, formats the time using the **timestamp** format. May only be used on keys whose type is `custom` or a time value. *Since SiLK 3.12.0.*

epoch

Print keys as seconds since UNIX epoch. May only be used on keys whose type is `custom` or a time value. *Since SiLK 3.12.0.*

--integer-keys

This switch is equivalent to **--key-format=decimal**, it is deprecated as of SiLK 3.7.0, and it will be removed in the SiLK 4.0 release.

--zero-pad-ips

This switch is equivalent to **--key-format=zero-padded**, it is deprecated as of SiLK 3.7.0, and it will be removed in the SiLK 4.0 release.

--no-columns

Disable fixed-width columnar output.

--column-separator=*C*

Use specified character between columns and after the final column. When this switch is not specified, the default of '|' is used.

--no-final-delimiter

Do not print the column separator after the final column. Normally a delimiter is printed. When the network summary is requested (**--network-structure=S**), the separator is always printed before the summary column and never after that column.

--delimited

--delimited=*C*

Run as if **--no-columns --no-final-delimiter --column-sep**=*C* had been specified. That is, disable fixed-width columnar output; if character *C* is provided, it is used as the delimiter between columns instead of the default '|'.

--output-path=*PATH*

Write the textual output of the **--network-structure**, **--bin-ips**, or **--sort-counters** switch to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output (and bypass the paging program). If *PATH* names an existing file, **rwbagcat** exits with an error unless the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. If this option is not given, the output is either sent to the pager or written to the standard output.

--pager=*PAGER_PROG*

When output is to a terminal, invoke the program *PAGER_PROG* to view the output one screen full at a time. This switch overrides the SILK_PAGER environment variable, which in turn overrides the PAGER variable. If the **--output-path** switch is given or if the value of the pager is determined to be the empty string, no paging is performed and all output is written to the terminal.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwbagcat** searches for the site configuration file in the locations specified in the FILES section. *Since SiLK 3.15.0.*

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line.

Printing a bag

To print the contents of the bag file *mybag.bag*:

```
$ rwbagcat mybag.bag
     172.23.1.1|              5|
     172.23.1.2|            231|
     172.23.1.3|              9|
     172.23.1.4|             19|
  192.168.0.100|              1|
  192.168.0.101|              1|
  192.168.0.160|             15|
 192.168.20.161|              1|
 192.168.20.162|              5|
 192.168.20.163|              5|
```

Displaying number of hosts by network

To print the bag with a full network breakdown:

```
$ rwbagcat --network-structure=TABCHX mybag.bag
          172.23.1.1      |              5|
          172.23.1.2      |            231|
          172.23.1.3      |              9|
          172.23.1.4      |             19|
        172.23.1.0/27     |            264|
      172.23.1.0/24       |            264|
    172.23.0.0/16         |            264|
  172.0.0.0/8             |            264|
          192.168.0.100   |              1|
          192.168.0.101   |              1|
        192.168.0.96/27   |              2|
          192.168.0.160   |             15|
        192.168.0.160/27  |             15|
      192.168.0.0/24      |             17|
          192.168.20.161  |              1|
          192.168.20.162  |              5|
          192.168.20.163  |              5|
        192.168.20.160/27 |             11|
      192.168.20.0/24     |             11|
    192.168.0.0/16        |             28|
  192.0.0.0/8             |             28|
TOTAL                     |            292|
```

In the above, lines that include a CIDR prefix display the sum of the preceding hosts. For example, there are 264 hosts in the 172.23.1.0/27 net-block.

To show an abbreviated network structure by class A and C only, including summary information:

```
$ rwbagcat --network-structure=ACS mybag.bag
    172.23.1.0/24     |            264| 4 hosts in 1 /27
```

```
172.0.0.0/8           |            264| 4 hosts in 1 /16, 1 /24, and 1 /27
    192.168.0.0/24    |             17| 3 hosts in 2 /27s
    192.168.20.0/24   |             11| 3 hosts in 1 /27
192.0.0.0/8           |             28| 6 hosts in 1 /16, 2 /24s, and 3 /27s
```

Overriding the key type

Suppose a key-type of a bag file is `duration`:

```
$ rwfileinfo --field=bag Bag2.bag
Bag2.bag:
  bag          key: duration @ 4 octets; counter: custom @ 8 octets
```

rwbagcat complains when the **--key-format** switch lists a format that it thinks is "nonsensical" for that type of key.

```
$ rwbagcat --key-format=utc Bag2.bag
rwbagcat: Invalid key-format 'utc':
       Nonsensical for Bag containing duration keys

$ rwbagcat --key-format=canonical Bag2.bag
rwbagcat: Invalid key-format 'canonical':
       Nonsensical for Bag containing duration keys
```

To use the **--key-format** one time and leave the key-type in the Bag file unchanged, you may merge the bag with an empty bag file: Use **rwbagbuild(1)** to create an empty bag that uses the `custom` key type, add the empty bag to *Bag2.bag* using **rwbagtool(1)**, then display the result:

```
$ rwbagbuild --bag-input=/dev/null   \
  | rwbagtool --add Bag2.bag stdin   \
  | rwbagcat --key-format=utc
1970/01/01T00:00:01|                  1|
1970/01/01T00:00:04|                  2|
1970/01/01T00:00:07|                 32|
1970/01/01T00:00:08|                  2|

$ rwbagbuild --bag-input=/dev/null   \
  | rwbagtool --add Bag2.bag -       \
  | rwbagcat --key-format=canonical
       0.0.0.1|                   1|
       0.0.0.4|                   2|
       0.0.0.7|                  32|
       0.0.0.8|                   2|
```

To rewrite the bag file with a different key type, print the bag file as text and use **rwbagbuild** to build a new bag file:

```
$ rwbagcat Bag2.bag    \
  | rwbagbuild --bag-input=- --key-type=sipv4
```

Inverting a bag

Inverting a bag means counting the number of times each counter appears in the bag.

To bin the number of IP addresses that had each flow count:

```
$ rwbagcat --bin-ips mybag.bag
               1|              3|
               5|              3|
               9|              1|
              15|              1|
              19|              1|
             231|              1|
```

The output shows that the bag contains 3 source hosts that had a single flow, 3 hosts that had 5 flows, and four hosts that each had a unique flow count (9, 15, 19, and 231).

For a log2 breakdown of the counts:

```
$ rwbagcat --bin-ips=binary mybag.bag
   2^0 to 2^1-1|              3|
   2^2 to 2^3-1|              3|
   2^3 to 2^4-1|              2|
   2^4 to 2^5-1|              1|
   2^7 to 2^8-1|              1|
```

Sorting the bag by counter value

rwbagcat normally presents the data in order of increasing key value. To sort based on the counter value, specify the **--sort-counter** switch. When sorting by the counter value, the default order is from maximum counter to minimum counter.

```
$ rwbagcat --sort-counter mybag.bag
     172.23.1.2|                231|
     172.23.1.4|                 19|
  192.168.0.160|                 15|
     172.23.1.3|                  9|
     172.23.1.1|                  5|
 192.168.20.162|                  5|
 192.168.20.163|                  5|
  192.168.0.100|                  1|
  192.168.0.101|                  1|
 192.168.20.161|                  1|
```

To change the sort order, specify the `increasing` argument to the **--sort-counter** switch:

```
$ rwbagcat --sort-counter=increasing mybag.bag
  192.168.0.100|                  1|
  192.168.0.101|                  1|
 192.168.20.161|                  1|
     172.23.1.1|                  5|
```

```
 192.168.20.162|                  5|
 192.168.20.163|                  5|
     172.23.1.3|                  9|
  192.168.0.160|                 15|
     172.23.1.4|                 19|
     172.23.1.2|                231|
```

For keys have the same counter value, the order of the keys is consistent (always from low to high) regardless how the counters are sorted. The following output is limited to those keys whose value is 5. The output is first shown without the **--sort-counter** switch, then with the data sorted by increasing and decreasing counter value.

```
$ rwbagcat --delim=, mybag.bag | grep ,5
172.23.1.1,5
192.168.20.162,5
192.168.20.163,5

$ rwbagcat --delim=, --sort-counter=increasing mybag.bag | grep ,5
172.23.1.1,5
192.168.20.162,5
192.168.20.163,5

$ rwbagcat --delim=, --sort-counter=decreasing mybag.bag | grep ,5
172.23.1.1,5
192.168.20.162,5
192.168.20.163,5
```

Displaying bags that use prefix map values as the key

rwbag(1) and **rwbagbuild(1)** can use a prefix map file as the key in a bag file as of SiLK 3.12.0. When attempting to display these Bag files, you must specify the **--pmap-file** switch on the **rwbagcat** command line for it to map each prefix map value to its label. If the **--pmap-file** is not given, **rwbagcat** displays an error.

```
$ rwbagcat service.bag
rwbagcat: The --pmap-file switch is required for \
        Bags containing sport-pmap keys
```

In addition, the type of the prefix map file must match the key-type in the bag file: a prefix map type of IPv4-address or IPv6-address when the key was mapped from an IP address, and a prefix map type of proto-port when the key was mapped from a protocol-port pair. The type of key in a bag may be determined by **rwfileinfo(1)**.

```
$ rwfileinfo --fields=bag service.bag
service.bag:
  bag          key: sport-pmap @ 4 octets; counter: custom @ 8 octets

$ rwbagcat --pmap-file=ip-map.pmap service.bag
rwbagcat: Cannot use IPv4-address prefix map for \
       Bag containing sport-pmap keys
```

```
$ rwbagcat --pmap-file=port-map.pmap service.bag
  TCP/SSH|                   1|
 TCP/SMTP|                 800|
 TCP/HTTP|                5642|
```

The only check **rwbagcat** makes is whether the prefix map file is the correct type. A different prefix map file may be used. If a value in the bag file does not have an index in the prefix map file, the numeric index of the label is displayed as shown in the following example which creates a prefix map with a single label.

```
$ echo 'label 1 none'                                          \
  | rwpmapbuild --mode=proto-port --input-path=-               \
      --output-path=tmp.pmap
$ rwbagcat --pmap-file=tmp.pmap service.bag
  7|                   1|
  8|                 800|
  9|                5642|
```

Displaying statistics

```
$ rwbagcat --print-statistics mybag.bag

Statistics
    number of keys:  10
   sum of counters:  292
       minimum key:  172.23.1.1
       maximum key:  192.168.20.163
   minimum counter:  1
   maximum counter:  231
              mean:  29.2
          variance:  5064
standard deviation:  71.16
              skew:  2.246
          kurtosis:  8.1
   nodes allocated:  0 (0 bytes)
   counter density:  inf%
```

ENVIRONMENT

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_PAGER

When set to a non-empty string, **rwbagcat** automatically invokes this program to display its output a screen at a time. If set to an empty string, **rwbagcat** does not automatically page its output.

PAGER

When set and SILK_PAGER is not set, **rwbagcat** automatically invokes this program to display its output a screen at a time.

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwbagcat** may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **rwbagcat** may use this environment variable. See the FILES section for details.

TZ

When the argument to the **--key-format** switch includes `localtime` or when a SiLK installation is built to use the local timezone, the value of the TZ environment variable determines the timezone in which **rwbagcat** displays timestamps. (If both of those are false, the TZ environment variable is ignored.) If the TZ environment variable is not set, the machine's default timezone is used. Setting TZ to the empty string or 0 causes timestamps to be displayed in UTC. For system information on the TZ variable, see **tzset(3)** or **environ(7)**. (To determine if SiLK was built with support for the local timezone, check the `Timezone support` value in the output of **rwbagcat --version**.)

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

SEE ALSO

rwbag(1), **rwbagbuild(1)**, **rwbagtool(1)**, **rwpmapbuild(1)**, **rwfileinfo(1)**, **rwset(1)**, **rwsetbuild(1)**, **silk(7)**, **tzset(3)**, **environ(7)**

rwbagtool

Perform high-level operations on binary Bag files

SYNOPSIS

```
rwbagtool { --add | --subtract | --minimize | --maximize
            | --divide | --scalar-multiply=VALUE
            | --compare={lt | le | eq | ge | gt} }
      [--intersect=SETFILE | --complement-intersect=SETFILE]
      [--mincounter=VALUE] [--maxcounter=VALUE]
      [--minkey=VALUE] [--maxkey=VALUE]
      [--invert] [--coverset] [--ipset-record-version=VERSION]
      [--output-path=PATH [--modify-inplace [--backup-path=BACKUP]]]
      [--note-strip] [--note-add=TEXT] [--note-file-add=FILE]
      [--compression-method=COMP_METHOD]
      [BAGFILE[ BAGFILE...]]

rwbagtool --help

rwbagtool --version
```

DESCRIPTION

rwbagtool performs various operations on binary Bag files (key-counter associations) and creates a new Bag file or an IPset file. **rwbagtool** can add Bags together, subtract a subset of data from a Bag, divide a Bag by another, compare the counters of two Bag files, perform key intersection of a Bag with an IPset, extract the keys of a Bag as an IPset, or filter Bag entries based on their key or counter values.

rwbagtool reads Bags from the files and named pipes specified on the command line. If no file names are given on the command line, **rwbagtool** attempts to read a Bag from the standard input. The names `stdin` or `-` may be used to force **rwbagtool** to read from the standard input. The resulting Bag or IPset is written to the location specified by the **--output-path** switch or to the standard output if that switch is not provided. If a *BAGFILE* does not contain a Bag or an attempt is made to read binary input or write binary output to the terminal,, **rwbagtool** prints an error to the standard error and exits abnormally.

In SiLK 3.21.0, **rwbagtool** added the **--modify-inplace** switch which correctly handles the case when an input file is also used as the output file. That switch causes **rwbagtool** to write the output to a temporary file first and then replace the original output file. The **--backup-path** switch may be used in conjunction with **--modify-inplace** to set the pathname where the original output file is copied.

A Bag is a set where each key is associated with a counter. **rwbag(1)** and **rwbagbuild(1)** are the primary tools used to create a Bag file. **rwbagcat(1)** prints a binary Bag file as text.

SiLK 3.15.0 introduced Aggregate Bags that are capable of storing multiple keys and counters. See **rwaggbag(1)**, **rwaggbagbuild(1)**, **rwaggbagcat(1)**, and **rwaggbagtool(1)** for more information.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required

for options that take optional parameters.

Operation switches

The first set of options are mutually exclusive; only one may be specified. If none are specified, the counters in the Bag files are summed.

--add

Sum the counters for each key for all Bag files given on the command line. At least one Bag file must be specified, and any number of additional Bag files may be given. If a key is not present in an input file, a counter of zero is used. The result contains the union of the keys from the input Bag files. When no operation switch is specified on the command line, the add operation is the default. If addition causes a counter to exceed the maximum value, **rwbagtool** exits with an error.

--subtract

Subtract from the first Bag file all subsequent Bag files. At least one Bag file must be specified, and any number of additional Bag files may be given. If a key does not appear in the first Bag file, **rwbagtool** assumes it has a value of 0. If subtracting a key's counters results in a non-positive number, the key does appear in the resulting Bag file. The result contains a subset of the keys in the first Bag file.

--minimize

Cause the output to contain the minimum counter seen for each key. Keys that do not appear in all input Bags do not appear in the output. At least one Bag file must be specified, and any number of additional Bag files may be given.

--maximize

Cause the output to contain the maximum counter seen for each key. The output contains each key that appears in any input Bag. At least one Bag file must be specified, and any number of additional Bag files may be given.

--divide

Divide the first Bag file by the second Bag file. It is an error if only one Bag file or more than two Bag files are given. Every key in the first Bag file must appear in the second file; the second Bag may have keys that do not appear in the first, and those keys do not appear in the output. Since Bags do not support floating point numbers, the result of the division is rounded to the nearest integer (values ending in `.5` are rounded up). If the result of the division is less than 0.5, the key does not appear in the output.

--scalar-multiply=*VALUE*

Multiply each counter in the Bag file by the scalar *VALUE*, where *VALUE* is an integer in the range 1 to 18446744073709551614. This switch requires a single Bag as input. On overflow, the lower 64-bits of the result are used as the counter's value.

--compare=*OPERATION*

Compare the key/counter pairs in exactly two Bag files. It is an error if only one Bag file or more than two Bag files are specified. The keys in the output Bag are only those for which the comparison denoted by *OPERATION* is true when comparing the key's counter in the first Bag with the key's counter in the second Bag. The counters for all keys in the output have the value 1. Any key that does not appear in both input Bag files does not appear in the result. The possible *OPERATION* values are the strings:

`lt`
: GetCounter(Bag1, key) < GetCounter(Bag2, key)

`le`
: GetCounter(Bag1, key) <= GetCounter(Bag2, key)

`eq`
: GetCounter(Bag1, key) == GetCounter(Bag2, key)

`ge`
: GetCounter(Bag1, key) >= GetCounter(Bag2, key)

`gt`
: GetCounter(Bag1, key) > GetCounter(Bag2, key)

Masking/Limiting switches

The result of the above operation is an intermediate Bag file. The following switches are applied next to remove entries from the intermediate Bag:

--intersect=*SETFILE*

Mask the keys in the intermediate Bag using the set in *SETFILE*. *SETFILE* is the name of a file or a named pipe containing an IPset, or the name `stdin` or `-` to have **rwbagtool** read the IPset from the standard input. If *SETFILE* does not contain an IPset, **rwbagtool** prints an error to stderr and exits abnormally. Only key/counter pairs where the key matches an entry in *SETFILE* are written to the output. (IPsets are typically created by **rwset(1)** or **rwsetbuild(1)**.)

--complement-intersect=*SETFILE*

As **--intersect**, but only writes key/counter pairs for keys which do **not** match an entry in *SETFILE*.

--mincounter=*VALUE*

Cause the output to contain only those entries whose counter value is *VALUE* or higher. The allowable range is 1 to the maximum counter value (18446744073709551614); the default is 1.

--maxcounter=*VALUE*

Cause the output to contain only those entries whose counter value is *VALUE* or lower. The allowable range is 1 to the maximum counter value; the default is the maximum counter value.

--minkey=*VALUE*

Cause the output to contain only those entries whose key value is *VALUE* or higher. Default is 0 (or 0.0.0.0). Accepts input as an integer or as an IP address in dotted decimal notation.

--maxkey=*VALUE*

Cause the output to contain only those entries whose key value is *VALUE* or higher. Default is 4294967295 (or 255.255.255.255). Accepts input as an integer or as an IP address in dotted decimal notation.

Output switches

The following switches control the output.

--invert

Generate a new Bag whose keys are the counters in the intermediate Bag and whose counter is the number of times the counter was seen. For example, this turns the Bag {sip:flow} into the Bag {flow:count(sip)}. Any counter in the intermediate Bag that is larger than the maximum possible key is attributed to the counter for the maximum key; to prevent this, specify `--maxcounter=4294967295` which removes all key-counter pairs whose counters do not fit into a key. (The **--bin-ips** switch on **rwbagcat(1)** allows one to invert a Bag file as it is being printed.) If inverting the Bag causes a counter to exceed the maximum value, **rwbagtool** exits with an error.

--coverset

Instead of creating a Bag file as the output, write an IPset which contains the keys contained in the intermediate Bag.

--ipset-record-version=*VERSION*

Specify the format of the IPset records that are written to the output when the **--coverset** switch is used. *VERSION* may be 2, 3, 4, 5 or the special value 0. When the switch is not provided, the SILK_IPSET_RECORD_VERSION environment variable is checked for a version. The default version is 0. *Since SiLK 3.11.0.*

0

Use the default version for an IPv4 IPset and an IPv6 IPset. Use the **--help** switch to see the versions used for your SiLK installation.

2

Create a file that may hold only IPv4 addresses and is readable by all versions of SiLK.

3

Create a file that may hold IPv4 or IPv6 addresses and is readable by SiLK 3.0 and later.

4

Create a file that may hold IPv4 or IPv6 addresses and is readable by SiLK 3.7 and later. These files are more compact that version 3 and often more compact than version 2.

5

Create a file that may hold only IPv6 addresses and is readable by SiLK 3.14 and later. When this version is specified, IPsets containing only IPv4 addresses are written in version 4. These files are usually more compact that version 4.

--output-path=*PATH*

Write the resulting Bag or IPset to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output. If *PATH* names an existing file, **rwbagtool** exits with an error unless the **--modify-inplace** switch is given or the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. If **--output-path** is not given, the output is written to the standard output. Attempting to write the binary output to a terminal causes **rwbagtool** to exit with an error.

--modify-inplace

Allow **rwbagtool** to overwrite an existing file and properly account for the output file (*PATH*) also being an input file. When this switch is given, **rwbagtool** writes the output to a temporary location first, then overwrites *PATH*. **rwbagtool** attempts to copy the permission, owner, and group from the original file to the new file. The switch is ignored when *PATH* does not exist or the output is the standard output or standard error. **rwbagtool** exits with an error when this switch is given and *PATH* is not a regular file. If **rwbagtool** encounters an error or is interrupted prior to closing the temporary file, the temporary file is removed. See also **--backup-path**. *Since SiLK 3.21.0.*

--backup-path=*BACKUP*

Move the file named by **--output-path** (*PATH*) to the path *BACKUP* immediately prior to moving the temporary file created by **--modify-inplace** over *PATH*. If *BACKUP* names a directory, the file is moved into that directory. This switch will overwrite an existing file. If *PATH* and *BACKUP* point to the same location, the output is written to *PATH* and no backup is created. If **BACKUP** cannot be created, the output is left in the temporary file and **rwbagtool** exits with a message and an error. **rwbagtool** exits with an error if this switch is given without **--modify-inplace**. *Since SiLK 3.21.0.*

--note-strip

Do not copy the notes (annotations) from the input files to the output file. Normally notes from the input files are copied to the output.

--note-add=*TEXT*

Add the specified *TEXT* to the header of the output file as an annotation. This switch may be repeated to add multiple annotations to a file. To view the annotations, use the **rwfileinfo(1)** tool.

--note-file-add=*FILENAME*

Open *FILENAME* and add the contents of that file to the header of the output file as an annotation. This switch may be repeated to add multiple annotations. Currently the application makes no effort to ensure that *FILENAME* contains text; be careful that you do not attempt to add a SiLK data file as an annotation.

--compression-method=*COMP_METHOD*

Specify the compression library to use when writing output files. If this switch is not given, the value in the SILK_COMPRESSION_METHOD environment variable is used if the value names an available compression method. When no compression method is specified, output to the standard output or to named pipes is not compressed, and output to files is compressed using the default chosen when SiLK was compiled. The valid values for *COMP_METHOD* are determined by which external libraries were found when SiLK was compiled. To see the available compression methods and the default method, use the **--help** or **--version** switch. SiLK can support the following *COMP_METHOD* values when the required libraries are available.

none

Do not compress the output using an external library.

zlib

Use the **zlib(3)** library for compressing the output, and always compress the output regardless of the destination. Using zlib produces the smallest output files at the cost of speed.

lzo1x

Use the *lzo1x* algorithm from the LZO real time compression library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead.

snappy

Use the *snappy* library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead. *Since SiLK 3.13.0.*

best

Use lzo1x if available, otherwise use snappy if available, otherwise use zlib if available. Only compress the output when writing to a file.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

The examples assume the following contents for the files:

```
Bag1.bag    Bag2.bag    Bag3.bag    Bag4.bag    Mask.set
 3|  10|     1|   1|     2|   8|     1|   1|          2
 4|   7|     4|   2|     4|  10|     4|   3|          4
 6|  14|     7|  32|     6|  14|     6|   4|          6
 7|  23|     8|   2|     7|  12|     7|   4|          8
 8|   2|                 9|   8|     8|   6|
```

The examples use **rwbagcat(1)** to print the contents of the Bag files.

Adding Bag Files

Adding Bag files produces a Bag whose keys are the set union of the keys in the input Bags. The counter for each key is the sum of the key's counters in each input Bag.

```
$ rwbagtool --add Bag1.bag Bag2.bag > Bag-sum.bag
$ rwbagcat --key-format=decimal Bag-sum.bag
 1|   1|
 3|  10|
 4|   9|
 6|  14|
 7|  55|
 8|   4|

$ rwbagtool --add Bag1.bag Bag2.bag Bag3.bag > Bag-sum2.bag
$ rwbagcat --key-format=decimal Bag-sum2.bag
 1|   1|
 2|   8|
 3|  10|
 4|  19|
 6|  28|
 7|  67|
 8|   4|
 9|   8|
```

Subtracting Bag Files

The **--subtract** switch subtracts from the key/counter pairs in the first Bag file the key/counter pairs in all other Bag file arguments. Keys that are not present in the first argument are ignored. If subtraction results in a counter value of zero or less, the key is removed from the result.

```
$ rwbagtool --subtract Bag1.bag Bag2.bag > Bag-diff.bag
$ rwbagcat --key-format=decimal Bag-diff.bag
 3|  10|
 4|   5|
 6|  14|

$ rwbagtool --subtract Bag2.bag Bag1.bag > Bag-diff2.bag
$ rwbagcat --key-format=decimal Bag-diff2.bag
 1|   1|
 7|   9|
```

Getting the Minimum Value

The output produced by the **--minimize** switch contains only the keys that appear in all of input Bags. For each key, the counter is the minimum value for that key in any input Bag.

```
$ rwbagtool --minimize Bag1.bag Bag2.bag Bag3.bag > Bag-min.bag
$ rwbagcat --key-format=decimal Bag-min.bag
 4|   2|
 7|  12|
```

Getting the Maximum Value

The keys of the Bag file produced by **--maximize** are the same as the keys produced by **--add**; that is, the union of all keys in the input files. For each key, its counter is the maximum value seen for that key in any single input Bag file.

```
$ rwbagtool --maximize Bag1.bag Bag2.bag Bag3.bag > Bag-max.bag
$ rwbagcat --key-format=decimal Bag-max.bag
 1|   1|
 2|   8|
 3|  10|
 4|  10|
 6|  14|
 7|  32|
 8|   2|
 9|   8|
```

Dividing Bag Files

The **--divide** switch requires exactly two Bag files as input. The keys in the first Bag argument must be either the same as or a subset of those in the second argument. The counter for each key in the first Bag file is divided by that key's counter in the second file. If the result of the division is less than 0.5, the key is not included in the output.

```
$ rwbagtool --divide Bag2.bag Bag4.bag > Bag-div1.bag
$ rwbagcat --key-format=decimal Bag-div1.bag
  1|   1|
  4|   1|
  7|   8|
```

When the order of the Bag file arguments is reversed an error is reported.

```
$ rwbagtool --divide Bag4.bag Bag2.bag > Bag-div2.bag
rwbagtool: Error dividing bags; key 6 not in divisor bag
```

To work around this issue, use the **--coverset** switch to create a copy of *Bag4.bag* that contains only the keys in *Bag2.bag*.

```
$ rwbagtool --coverset Bag2.bag > Bag2-keys.set
$ rwbagtool --intersect=Bag2-keys.set  Bag4.bag > Bag4-small.bag
$ rwbagtool --divide Bag4-small.bag Bag2.bag > Bag-div2.bag
$ rwbagcat --key-format=decimal Bag-div2.bag
  1|   1|
  4|   2|
  8|   3|
```

The following command is the same as the above except the IPset and Bag files are piped between the tools instead of being written to disk:

```
$ rwbagtool --coverset Bag2.bag                \
  | rwbagtool --intersect=-  Bag4.bag          \
  | rwbagtool --divide -  Bag2.bag             \
  | rwbagcat --key-format=decimal
  1|   1|
  4|   2|
  8|   3|
```

Scalar Multiplication

The **--scalar-multiply** switch multiplies each counter in the input Bag by the specified value. Exactly one Bag file argument is required.

```
$ rwbagtool --scalar-multiply=7 Bag1.bag > Bag-multiply.bag
$ rwbagcat --key-format=decimal Bag-multiply.bag
 3|  70|
 4|  49|
 6|  98|
 7| 161|
 8|  14|
```

Use two **rwbagtool** commands if multiple operations are desired.

```
$ rwbagtool --add Bag1.bag Bag2.bag   \
  | rwbagtool --scalar-multiply=3 --output-path=Bag12-multi.bag
$ rwbagcat --key-format=decimal Bag12-multi.bag
 1|   3|
 3|  30|
 4|  27|
 6|  42|
 7| 165|
 8|  12|
```

Comparing Bag Files

The **--compare** switch takes an argument that specifies how to compare the counters in two Bag files, and it requires exactly two Bag files as input. For each key that appears in both Bag files, the counter value in the first file is compared to counter value in the second file. If the comparison is true, the key appears in the resulting Bag file with a counter of 1. If the comparison is false, the key is not present in the output file. Keys that appear in only one of the input files are ignored.

The following comparisons operate on *Bag1.bag* and *Bag2.bag* which have as common keys 4, 7, and 8.

Find counters in *Bag1.bag* that are less than those in *Bag2.bag*:

```
$ rwbagtool --compare=lt Bag1.bag Bag2.bag > Bag-lt.bag
$ rwbagcat --key-format=decimal Bag-lt.bag
 7|   1|
```

Find counters in *Bag1.bag* that are less than or equal to those in *Bag2.bag*:

```
$ rwbagtool --compare=le Bag1.bag Bag2.bag > Bag-le.bag
$ rwbagcat --key-format=decimal Bag-le.bag
 7|   1|
 8|   1|
```

Find counters in *Bag1.bag* that are equal to those in *Bag2.bag*:

```
$ rwbagtool --compare=eq Bag1.bag Bag2.bag > Bag-eq.bag
$ rwbagcat --key-format=decimal Bag-eq.bag
 8|   1|
```

Find counters in *Bag1.bag* that are greater than or equal to those in *Bag2.bag*:

```
$ rwbagtool --compare=ge Bag1.bag Bag2.bag > Bag-ge.bag
$ rwbagcat --key-format=decimal Bag-ge.bag
 4|   1|
 8|   1|
```

Find counters in *Bag1.bag* that are greater than those in *Bag2.bag*:

```
$ rwbagtool --compare=gt Bag1.bag Bag2.bag > Bag-gt.bag
$ rwbagcat --key-format=decimal Bag-gt.bag
 4|   1|
```

Making a Cover Set

A *cover set* is an IPset file that contains the keys that are present in any of the input Bag files. In other words, it is the union of the keys converted to an IPset. Since an operation switch is not provided in this command, an implicit **--add** operation is performed on the Bag files prior to creating the cover set. (**rwsetcat(1)** prints the contents of an IPset file as text.)

```
$ rwbagtool --coverset Bag1.bag Bag2.bag Bag3.bag > Cover.set
$ rwsetcat --key-format=decimal Cover.set
 1
 2
 3
 4
 6
 7
 8
 9
```

One use of a cover set is to limit the contents of a Bag file to keys that are present in a second Bag file:

```
$ rwbagtool --coverset --output-path=Cover.set Bag1.bag
$ rwbagtool --intersect=Cover.set Bag2.bag > Bag1-mask-Bag2.bag
$ rwbagcat --key-format=decimal Bag1-mask-Bag2.bag
 4|   2|
 7|  32|
 8|   2|
```

To mask the contents of *Bag2.bag* by the keys that are not present in *Bag1.bag*:

```
$ rwbagtool --complement-intersect=Cover.set Bag2.bag \
       > Bag1-notmask-Bag2.bag
$ rwbagcat --key-format=decimal Bag1-notmask-Bag2.bag
 1|   1|
```

Inverting a Bag

The output of the **--invert** switch is a Bag file that counts the number of times each counter is present in the input Bag file.

```
$ rwbagtool --invert Bag1.bag > Bag-inv1.bag
$ rwbagcat --key-format=decimal Bag-inv1.bag
 2|   1|
 7|   1|
10|   1|
14|   1|
23|   1|
```

```
$ rwbagtool --invert Bag2.bag > Bag-inv2.bag
$ rwbagcat --key-format=decimal Bag-inv2.bag
 1|   1|
 2|   2|
32|   1|
```

```
$ rwbagtool --invert Bag3.bag > Bag-inv3.bag
$ rwbagcat --key-format=decimal Bag-inv3.bag
 8|   2|
```

```
 10|   1|
 12|   1|
 14|   1|
```

When multiple Bag files are specified on the command line, the files are added prior to creating the inverted Bag. Even though the counter 2 appears three times in the files *Bag1.bag* and *Bag2.bag*, the key 2 is not present in the following since the add operation is performed first.

```
$ rwbagtool --invert Bag1.bag Bag2.bag   \
  | rwbagcat --key-format=decimal
 1|   1|
 4|   1|
 9|   1|
10|   1|
14|   1|
55|   1|
```

Masking Bag Files

The **--intersect** switch takes an IPset file as an argument and limits the keys of the Bag produced by **rwbagtool** to only those keys that appear in the IPset file.

```
$ rwbagtool --intersect=Mask.set Bag1.bag > Bag-mask.bag
$ rwbagcat --key-format=decimal Bag-mask.bag
 4|   7|
 6|  14|
 8|   2|
```

The **--complement-intersect** switch limits the output to only those keys that do not appear in the IPset file.

```
$ rwbagtool --complement-intersect=Mask.set Bag1.bag > Bag-mask2.bag
$ rwbagcat --key-format=decimal Bag-mask2.bag
 3|  10|
 7|  23|
```

See also the next section.

Restricting the Output

In addition to limiting the result of **rwbagtool** to keys that appear or do not appear in an IPset file (cf. previous section), numeric limits may be used to restrict the keys or counters that in the resulting Bag file with use of the **--minkey**, **--maxkey**, **--mincounter**, and **--maxcounter** switches.

```
$ rwbagtool --add --maxkey=5 Bag1.bag Bag2.bag > Bag-res1.bag
$ rwbagcat --key-format=decimal Bag-res1.bag
 1|   1|
 3|  10|
 4|   9|
```

```
$ rwbagtool --minkey=3 --maxkey=6 Bag1.bag > Bag-res2.bag
$ rwbagcat --key-format=decimal Bag-res2.bag
 3|  10|
 4|   9|
 6|  14|

$ rwbagtool --mincounter=20 Bag1.bag Bag2.bag > Bag-res3.bag
$ rwbagcat --key-format=decimal Bag-res3.bag
 7|  55|

$ rwbagtool --subtract --maxcounter=9 Bag1.bag Bag2.bag  \
      > Bag-res4.bag
$ rwbagcat --key-format=decimal Bag-res4.bag
 4|   5|
```

Changing a File's Format

To share a Bag file with a user who has a version of SiLK that includes different compression libraries, it may be necessary to change the the compression-method of the Bag.

It is not possible to change the compression-method directly. A new file must be created first, and then you may then replace the old file with the new file.

To create a new file that uses a different compression-method of the Bag file *A.bag*, use **rwbagtool** with the **--add** switch and specify the desired argument:

```
$ rwbagtool --add --compression=none --output-path=A1.bag A.bag
```

Changing the Key Type or Counter Type

Unfortunately, the Bag tools do not allow changing the key type or counter type of a Bag file. To change the types, use **rwbagcat(1)** to write the Bag as text and **rwbagbuild(1)** to convert the text back to a Bag file.

```
$ rwbagcat Bag1.bag    \
  | rwbagbuild --bag-input=- --output-path=Bag1-typed.bag  \
       --key-type=sport --counter-type=sum-bytes
```

Use **rwfileinfo(1)** to see the type of the key and counter.

```
$ rwfileinfo --field=bag Bag1-typed.bag
Bag1-typed.bag:
  bag          key: sPort @ 4 octets; counter: sum-bytes @ 8 octets
```

Alternatively, one may use PySiLK (see pysilk(3)) to modify the key type and counter type.

```
$ cat bag-type.py
import sys
from silk import *
```

```
key_type = sys.argv[1]
counter_type = sys.argv[2]
old_file = sys.argv[3]
new_file = sys.argv[4]

old = Bag.load(old_file, key_type=IPv4Addr)
new = Bag(old, key_type=key_type, counter_type=counter_type)
new.save(new_file)
$
$ python bag-type.py sipv4 sum-packets Bag1.bag Bag1-type2.bag
$ rwfileinfo --field=bag Bag1-type2.bag
Bag1-type2.bag:
  bag          key: sIPv4 @ 4 octets; counter: sum-packets @ 8 octets
```

ENVIRONMENT

SILK_IPSET_RECORD_VERSION

This environment variable is used as the value for the **--ipset-record-version** when that switch is not provided. *Since SiLK 3.7.0.*

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_COMPRESSION_METHOD

This environment variable is used as the value for **--compression-method** when that switch is not provided. *Since SiLK 3.13.0.*

NOTES

The **--modify-inplace** switch was added in SiLK 3.21. When **--backup-path** is also given, there is a small time window when the original file does not exist: the time between moving the original file to the backup location and moving the temporary file into place.

rwbagtool should handle counter overflow more consistently and gracefully.

SEE ALSO

rwbag(1), **rwbagbuild(1)**, **rwbagcat(1)**, **rwfileinfo(1)**, **rwset(1)**, **rwsetbuild(1)**, **rwsetcat(1)**, **rwaggbag(1)**, **rwaggbagbuild(1)**, **rwaggbagcat(1)**, **rwaggbagtool(1)**, **silk(7)**, **zlib(3)**

rwcat

Concatenate SiLK Flow files into single stream

SYNOPSIS

```
rwcat [--output-path=PATH] [--note-add=TEXT] [--note-file-add=FILE]
      [--print-filenames] [--byte-order={big | little | native}]
      [--ipv4-output] [--compression-method=COMP_METHOD]
      [--site-config-file=FILENAME]
      {[--xargs] | [--xargs=FILENAME] | [FILE [FILE...]]}

rwcat --help

rwcat --version
```

DESCRIPTION

rwcat reads SiLK Flow records and writes the records in the standard binary SiLK format to the specified output-path; **rwcat** writes the records to the standard output when stdout is not the terminal and **--output-path** is not provided.

rwcat reads SiLK Flow records from the files named on the command line or from the standard input when no file names are specified and **--xargs** is not present. To read the standard input in addition to the named files, use `-` or `stdin` as a file name. If an input file name ends in `.gz`, the file is uncompressed as it is read. When the **--xargs** switch is provided, **rwcat** reads the names of the files to process from the named text file or from the standard input if no file name argument is provided to the switch. The input to **--xargs** must contain one file name per line.

rwcat does *not* copy the invocation history and annotations (notes) from the header(s) of the source file(s) to the destination file. The **--note-add** or **--note-file-add** switch may be used to add a new annotation to the destination file.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--output-path=*PATH*

Write the binary SiLK Flow records to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output. If *PATH* names an existing file, **rwcat** exits with an error unless the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. When *PATH* ends in `.gz`, the output is compressed using the library associated with **gzip(1)**. If this switch is not given, the output is written to the standard output. Attempting to write the binary output to a terminal causes **rwcat** to exit with an error.

--note-add=*TEXT*

Add the specified *TEXT* to the header of the output file as an annotation. This switch may be repeated to add multiple annotations to a file. To view the annotations, use the **rwfileinfo(1)** tool.

--note-file-add=*FILENAME*

Open *FILENAME* and add the contents of that file to the header of the output file as an annotation. This switch may be repeated to add multiple annotations. Currently the application makes no effort to ensure that *FILENAME* contains text; be careful that you do not attempt to add a SiLK data file as an annotation.

--byte-order=*ENDIAN*

Set the byte order for the output SiLK Flow records. The argument is one of the following:

`native`
: Use the byte order of the machine where **rwcat** is running. This is the default.

`big`
: Use network byte order (big endian) for the output.

`little`
: Write the output in little endian format.

--ipv4-output

Force the output to contain only IPv4 flow records. When this switch is specified, IPv6 flow records that contain addresses in the ::ffff:0:0/96 prefix are converted to IPv4 and written to the output, and all other IPv6 records are ignored. When SiLK has not been compiled with IPv6 support, **rwcat** acts as if this switch were always in effect.

--compression-method=*COMP_METHOD*

Specify the compression library to use when writing output files. If this switch is not given, the value in the SILK_COMPRESSION_METHOD environment variable is used if the value names an available compression method. When no compression method is specified, output to the standard output or to named pipes is not compressed, and output to files is compressed using the default chosen when SiLK was compiled. The valid values for *COMP_METHOD* are determined by which external libraries were found when SiLK was compiled. To see the available compression methods and the default method, use the **--help** or **--version** switch. SiLK can support the following *COMP_METHOD* values when the required libraries are available.

none
: Do not compress the output using an external library.

zlib
: Use the **zlib(3)** library for compressing the output, and always compress the output regardless of the destination. Using zlib produces the smallest output files at the cost of speed.

lzo1x
: Use the *lzo1x* algorithm from the LZO real time compression library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead.

snappy
: Use the *snappy* library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead. *Since SiLK 3.13.0.*

best

Use lzo1x if available, otherwise use snappy if available, otherwise use zlib if available. Only compress the output when writing to a file.

--print-filenames

Print the names of input files and the number of records each file contains as the files are read.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwcat** searches for the site configuration file in the locations specified in the FILES section.

--xargs

--xargs=*FILENAME*

Read the names of the input files from *FILENAME* or from the standard input if *FILENAME* is not provided. The input is expected to have one filename per line. **rwcat** opens each named file in turn and reads records from it as if the filenames had been listed on the command line.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

To combine the results of several **rwfilter(1)** runs---stored in the files *run1.rw*, *run2.rw*, ... *runN.rw*---together to create the file *combined.rw*, you can use:

```
$ rwcat --output=combined.rw  *.rw
```

If the shell complains about too many arguments, you can use the UNIX **find(1)** function and pipe its output to **rwcat**:

```
$ find . -name '*.rw' -print                    \
  | rwcat --xargs --output=combined.rw
```

ENVIRONMENT

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_COMPRESSION_METHOD

This environment variable is used as the value for **--compression-method** when that switch is not provided. *Since SiLK 3.13.0.*

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwcat** may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **rwcat** may use this environment variable. See the FILES section for details.

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

SEE ALSO

rwfilter(1), **rwfileinfo(1)**, **silk(7)**, **gzip(1)**, **find(1)**, **zlib(3)**

BUGS

Although **rwcat** will read from the standard input, this feature should be used with caution. **rwcat** will treat the standard input as a single file, as it has no way to know when one file ends and the next begins. The following will not work:

```
$ cat run1.rw run2.rw | rwcat --output=combined.rw     # WRONG!
```

The header of *run2.rw* will be treated as data of *run1.rw*, resulting in corrupt output.

rwcombine

Combine flows denoting a long-lived session into a single flow

SYNOPSIS

```
rwcombine [--actions=ACTIONS] [--ignore-fields=FIELDS]
      [--max-idle-time=NUM]
      [{--print-statistics | --print-statistics=FILENAME}]
      [--temp-directory=DIR_PATH] [--buffer-size=SIZE]
      [--note-add=TEXT] [--note-file-add=FILE]
      [--compression-method=COMP_METHOD] [--print-filenames]
      [--output-path=PATH] [--site-config-file=FILENAME]
      {[--xargs] | [--xargs=FILENAME] | [FILE [FILE ...]]}

rwcombine --help

rwcombine --help-fields

rwcombine --version
```

DESCRIPTION

rwcombine reads SiLK Flow records from one or more input sources, searches for flow records where the *attributes* field denotes records that were prematurely created or were continuations of prematurely created flows, and attempts to combine those records into a single record. All the unmodified SiLK records and the combined records are written to the file specified by the **--output-path** switch or to the standard output when the **--output-path** switch is not provided and the standard output is not connected to a terminal.

Some flow exporters, such as **yaf(1)**, provide fields that describe characteristics about the flow record, and these characteristics are stored in the *attributes* field of SiLK Flow records. The two flags that **rwcombine** considers are:

`T`

> The flow generator prematurely created a record for a long-lived session due to the connection's lifetime reaching the *active timeout* of the flow generator. (Also, when **yaf** is run with the **--silk** switch, it prematurely creates a flow and marks it with `T` if the byte count of the flow cannot be stored in a 32-bit value.)

`C`

> The flow generator created this flow as a continuation of long-running connection, where the previous flow for this connection met a timeout. (**yaf** only sets this flag when it is invoked with the **--silk** switch.)

A very long-running session may be represented by multiple flow records, where the first record is marked with the `T` flag, the final record is marked with the `C` flag, and intermediate records are marked with both `C` (this record continues an earlier flow) and `T` (this record also met the active time-out). **rwcombine** attempts to combine these multiple flow records into a single record.

The input to **rwcombine** does not need to be sorted. As part of its processing, **rwcombine** may re-order the records before writing them.

rwcombine reads SiLK Flow records from the files named on the command line or from the standard input when no file names are specified and **--xargs** is not present. To read the standard input in addition to the named files, use `-` or `stdin` as a file name. If an input file name ends in `.gz`, the file is uncompressed as it is read. When the **--xargs** switch is provided, **rwcombine** reads the names of the files to process from the named text file or from the standard input if no file name argument is provided to the switch. The input to **--xargs** must contain one file name per line.

Algorithm

The algorithm **rwcombine** uses to combine records is

1. **rwcombine** reads SiLK flow records, examines the *attributes* field on each record, and immediately writes to the destination stream all records where both the time-out flag (`T`) and the continuation flag (`C`) are not set. Records where one or both of those flags are set are stored until all input records have been read.
2. **rwcombine** groups the stored records into bins where the following fields for each record in each bin are identical: *sIP*, *dIP*, *sPort*, *dPort*, *protocol*, *sensor*, *in*, *out*, *nhIP*, *application*, *class*, and *type*.
3. For each bin, the records are stored by time (*sTime* and *elapsed*).
4. Within a bin, **rwcombine** combines two records into a single record when the *attributes* field of the first record has the `T` (time-out) flag set and the second record has the `C` (continuation) flag set. When combining records, the *bytes* field and *packets* fields are summed, the *initialFlags* from the first record is used, the *sessionFlags* field becomes the bit-wise OR of both *sessionFlags* fields and the second record's *initialFlags* field, and the *eTime* is set to that of the second flow.
5. If the second record's `T` flag was set, **rwcombine** checks to see if the third record's `C` flag is set. If it is, the third record becomes part of the new record.
6. The previous step repeats for the records in the bin until the bin contains a single record, the most recently added record did not have the `T` flag set, or the next record in the bin does not have the `C` flag set.
7. After examining a bin, **rwcombine** writes the record(s) the bin contains to the destination stream.
8. Steps 3 through 7 are repeated for each bin.

The **--ignore-fields** switch allows the user to remove fields from the set that **rwcombine** uses when grouping records in Step 2.

When combining two records into one (Step 4), **rwcombine** completely disregards the difference between the first record's end-time and the second record's start-time (the *idle time*). To tell **rwcombine** not to combine those records when the difference is greater than a limit, specify that value as the argument to the **--max-idle-time** switch.

To see information on the number of flows combined and the minimum and maximum idle times, specify the **--print-statistics** switch.

During its processing, **rwcombine** will try to allocate a large (near 2GB) in-memory array to hold the records. (You may use the **--buffer-size** switch to change this maximum buffer size.) If more records are read than will fit into memory, the in-core records are temporarily stored on disk as described by the

--temp-directory switch. When all records have been read, the on-disk files are merged to produce the output.

By default, the temporary files are stored in the */tmp* directory. Because the sizes of the temporary files may be large, it is strongly recommended that */tmp not* be used as the temporary directory, and **rwcombine** will print a warning when */tmp* is used. To modify the temporary directory used by **rwcombine**, provide the **--temp-directory** switch, set the SILK_TMPDIR environment variable, or set the TMPDIR environment variable.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--actions=*ACTIONS*

Select the type of action(s) that **rwcombine** should take to combine the input records. The default action is `all`, and the following actions are supported:

all

Perform all the actions described below.

timeout

Combine into a single flow record those records where the timeout flags in the *attributes* field indicate that the flow exporter has divided a long-lived session into multiple flow records.

This switch is provided for future expansion of **rwcombine**, since at present **rwcombine** supports a single action. When writing a script that uses **rwcombine**, specify **--action=timeout** for compatibility with future versions of **rwcombine**.

--ignore-fields=*FIELDS*

Ignore the fields listed in *FIELDS* when determining if two flow records should be grouped into the same bin; that is, treat *FIELDS* as being identical across all flows. By default, **rwcombine** puts records into a bin when the records have identical values for the following fields: sIP, dIP, sPort, dPort, protocol, sensor, in, out, nhIP, application, class, and type.

FIELDS is a comma separated list of field-names, field-integers, and ranges of field-integers; a range is specified by separating the start and end of the range with a hyphen (**-**). Field-names are case-insensitive. Example:

```
--ignore-fields=sensor,12-15
```

The list of supported fields are:

sIP,1

source IP address

dIP,2

destination IP address

sPort,3

source port for TCP and UDP, or equivalent

dPort,4
destination port for TCP and UDP, or equivalent

protocol,5
IP protocol

sensor,12
name or ID of sensor at the collection point

in,13
router SNMP input interface or vlanId if packing tools were configured to capture it (see **sensor.conf(5)**)

out,14
router SNMP output interface or postVlanId

nhIP,15
router next hop IP

class,20,type,21
class and type of sensor at the collection point (represented internally by a single value)

application,29
guess as to the content of the flow. Some software that generates flow records from packet data, such as **yaf(1)**, will inspect the contents of the packets that make up a flow and use traffic signatures to label the content of the flow. SiLK calls this label the *application*; **yaf** refers to it as the *appLabel*. The application is the port number that is traditionally used for that type of traffic (see the */etc/services* file on most UNIX systems). For example, traffic that the flow generator recognizes as FTP will have a value of 21, even if that traffic is being routed through the standard HTTP/web port (80).

--max-idle-time=*NUM*
Do not combine flow records when the start time of the second flow record begins *NUM* seconds after the end time of the first flow record. *NUM* may be fractional. If not specified, the maximum idle time may be considered infinite.

--print-statistics

--print-statistics=*FILENAME*
Print to the standard error or to the specified *FILENAME* the number of flows records read and written, the number of flows that did not require combining, the number of flows combined, the number that could not be combined, and minimum and maximum idle time between combined flow records.

--temp-directory=*DIR_PATH*
Specify the name of the directory in which to store data files temporarily when more records have been read that will fit into RAM. This switch overrides the directory specified in the SILK_TMPDIR environment variable, which overrides the directory specified in the TMPDIR variable, which overrides the default, */tmp*.

--buffer-size=*SIZE*
Set the maximum size of the buffer to use for holding the records, in bytes. A larger buffer means fewer temporary files need to be created, reducing the I/O wait times. The default maximum for this buffer is near 2GB. The *SIZE* may be given as an ordinary integer, or as a real number followed by a suffix `K`, `M` or `G`, which represents the numerical value multiplied by 1,024 (kilo), 1,048,576 (mega), and 1,073,741,824 (giga), respectively. For example, 1.5K represents 1,536 bytes, or one and one-half kilobytes. (This value does **not** represent the absolute maximum amount of RAM that **rwcombine** will allocate, since additional buffers will be allocated for reading the input and writing the output.)

--output-path=*PATH*

Write the binary SiLK Flow records to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output. If *PATH* names an existing file, **rwcombine** exits with an error unless the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. If this switch is not given, the output is written to the standard output. Attempting to write the binary output to a terminal causes **rwcombine** to exit with an error.

--note-add=*TEXT*

Add the specified *TEXT* to the header of the output file as an annotation. This switch may be repeated to add multiple annotations to a file. To view the annotations, use the **rwfileinfo(1)** tool.

--note-file-add=*FILENAME*

Open *FILENAME* and add the contents of that file to the header of the output file as an annotation. This switch may be repeated to add multiple annotations. Currently the application makes no effort to ensure that *FILENAME* contains text; be careful that you do not attempt to add a SiLK data file as an annotation.

--compression-method=*COMP_METHOD*

Specify the compression library to use when writing output files. If this switch is not given, the value in the SILK_COMPRESSION_METHOD environment variable is used if the value names an available compression method. When no compression method is specified, output to the standard output or to named pipes is not compressed, and output to files is compressed using the default chosen when SiLK was compiled. The valid values for *COMP_METHOD* are determined by which external libraries were found when SiLK was compiled. To see the available compression methods and the default method, use the **--help** or **--version** switch. SiLK can support the following *COMP_METHOD* values when the required libraries are available.

none

Do not compress the output using an external library.

zlib

Use the **zlib(3)** library for compressing the output, and always compress the output regardless of the destination. Using zlib produces the smallest output files at the cost of speed.

lzo1x

Use the *lzo1x* algorithm from the LZO real time compression library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead.

snappy

Use the *snappy* library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead. *Since SiLK 3.13.0.*

best

Use lzo1x if available, otherwise use snappy if available, otherwise use zlib if available. Only compress the output when writing to a file.

--print-filenames

Print to the standard error the names of input files as they are opened.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwcombine** searches for the site configuration file in the locations specified in the FILES section.

--xargs

--xargs=*FILENAME*

Read the names of the input files from *FILENAME* or from the standard input if *FILENAME* is not provided. The input is expected to have one filename per line. **rwcombine** opens each named file in turn and reads records from it as if the filenames had been listed on the command line.

--help

Print the available options and exit.

--help-fields

Print the description and alias(es) of each field and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

Use **rwfilter(1)** to find **ssh** flow records that involve the host 192.168.126.252. The output from **rwcut(1)** shows the flow exporter split this long-lived **ssh** session into multiple flow records:

```
$ rwfilter --saddr=192.168.126.252 --dport=22 --pass=- data.rw \
  | rwcut --fields=flags,attributes,stime,etime
   flags|attribut|                  sTime|                  eTime|
 S PA   |T       |2009/02/13T00:29:59.563|2009/02/13T00:59:39.668|
   PA   |TC      |2009/02/13T00:59:39.668|2009/02/13T01:29:19.478|
   PA   |TC      |2009/02/13T01:29:19.478|2009/02/13T01:58:48.890|
   PA   |TC      |2009/02/13T01:58:48.891|2009/02/13T02:28:43.599|
F  PA   | C      |2009/02/13T02:28:43.600|2009/02/13T02:32:58.272|
```

Here is the other half of that conversation:

```
$ rwfilter --daddr=192.168.126.252 --sport=22 --pass=- data.rw \
  | rwcut --fields=flags,attributes,stime,etime
   flags|attribut|                  sTime|                  eTime|
 S PA   |T       |2009/02/13T00:30:00.060|2009/02/13T00:59:39.667|
   PA   |TC      |2009/02/13T00:59:39.670|2009/02/13T01:29:19.478|
   PA   |TC      |2009/02/13T01:29:19.481|2009/02/13T01:58:48.890|
   PA   |TC      |2009/02/13T01:58:48.893|2009/02/13T02:28:43.599|
F  PA   | C      |2009/02/13T02:28:43.600|2009/02/13T02:32:58.271|
```

Use **rwuniq(1)** to compute the byte and packet counts for that **ssh** session:

```
$ rwfilter --any-addr=192.168.126.252 --aport=22 --pass=- data.rw \
  | rwuniq --fields=sip,dip,sport,dport --values=records,byte,packets
           sIP|            dIP|sPort|dPort|Records|  Bytes|Packets|
  10.11.156.107|192.168.126.252|   22|28975|      5|4677240|   3881|
192.168.126.252|  10.11.156.107|28975|   22|      5| 281939|   3891|
```

Invoke **rwcombine** on these records and store the result in the file *combined.rw*:

```
$ rwfilter --any-addr=192.168.126.252 --aport=22 --pass=- data.rw \
  | rwcombine --print-statistics --output-path=combined.rw
FLOW RECORD COUNTS:
Read:                                  10
Initially Complete:           -         0 *
Sorted & Examined:            =        10
Missing end:                  -         0 *
Missing start & end:          -         0 *
Missing start:                -         0 *
Prior to combining:           =        10
Eliminated:                   -         8
Made complete:                =         2 *
Written:                                2 (sum of *)

IDLE TIMES:
Minimum:        0:00:00:00.000
Penultimate:    0:00:00:00.000
Maximum:        0:00:00:00.003
```

View the resulting records:

```
$ rwcut --fields=sip,dip,sport,dport,bytes,packets,flags combined.rw
            sIP|            dIP|sPort|dPort|  bytes|packets|   flags|
  10.11.156.107|192.168.126.252|   22|28975|4677240|   3881|FS PA   |
192.168.126.252|  10.11.156.107|28975|   22| 281939|   3891|FS PA   |

$ rwcut --fields=sip,attributes,stime,etime combined.rw
            sIP|attribut|                  sTime|                  eTime|
  10.11.156.107|        |2009/02/13T00:30:00.060|2009/02/13T02:32:58.271|
192.168.126.252|        |2009/02/13T00:29:59.563|2009/02/13T02:32:58.272|
```

ENVIRONMENT

SILK_TMPDIR

When set and **--temp-directory** is not specified, **rwcombine** writes the temporary files it creates to this directory. SILK_TMPDIR overrides the value of TMPDIR.

TMPDIR

When set and SILK_TMPDIR is not set, **rwcombine** writes the temporary files it creates to this directory.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_COMPRESSION_METHOD

This environment variable is used as the value for **--compression-method** when that switch is not provided. *Since SiLK 3.13.0.*

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwcombine** may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **rwcombine** may use this environment variable. See the FILES section for details.

SILK_TEMPFILE_DEBUG

When set to 1, **rwcombine** prints debugging messages to the standard error as it creates, re-opens, and removes temporary files.

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

${SILK_TMPDIR}/

${TMPDIR}/

/tmp/

Directory in which to create temporary files.

SEE ALSO

rwfilter(1), **rwcut(1)**, **rwuniq(1)**, **rwfileinfo(1)**, **sensor.conf(5)**, **silk(7)**, **yaf(1)**, **zlib(3)**

NOTES

The first release of **rwcombine** occurred in SiLK 3.9.0.

rwcompare

Compare the records in two SiLK Flow files

SYNOPSIS

```
rwcompare [--quiet] [--site-config-file] FILE1 FILE2

rwcompare --help

rwcompare --version
```

DESCRIPTION

rwcompare opens the two files named on the command and compares the SiLK Flow records they contain. If the records are identical, **rwcompare** exits with status 0. If any of the records differ, **rwcompare** prints a message and exits with status 1. If there is an issue reading either file, an error is printed and the exit status is 2. Use the **--quiet** switch to suppress all output (error messages included). You may use `-` or `stdin` for one of the file names, in which case **rwcompare** reads from the standard input.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--quiet

Do not print a message if the files differ, and do not an print error message if a file cannot be opened or read.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwcombine** searches for the site configuration file in the locations specified in the FILES section.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (**$**) represents the shell prompt. Some input lines are split over multiple lines in order to improve readability, and a backslash (\) is used to indicate such lines. The examples assume the existence of the file *data.rw* that contains SiLK Flow records. The exit status of the most recent command is available in the shell variable `$?`.

Compare a file with itself:

```
$ rwcompare data.rw data.rw
$ echo $?
0
```

Compare a file with itself, where one instance of the file is read from the standard input:

```
$ rwcat data.rw | rwcompare - data.rw
$ echo $?
0
```

Use **rwsort(1)** to modify one instance of the file and compare the results:

```
$ rwsort --fields=proto data.rw | rwcompare - data.rw
- data.rw differ: record 1
$ echo $?
1
```

Run the command again and use the **--quiet** switch:

```
$ rwsort --fields=proto data.rw | rwcompare --quiet - data.rw
$ echo $?
1
```

Compare the file with input containing two copies of the file:

```
$ rwcat data.rw data.rw | rwcompare data.rw -
data.rw - differ: EOF data.rw
$ echo $?
1
```

Compare the file with */dev/null*:

```
$ rwcompare --quiet /dev/null data.rw
$ echo $?
2
```

rwcompare checks whether two files have the same records in the same order. To compare two arbitrary files, use **rwsort(1)** to reorder the records. Make certain to provide enough fields to the **rwsort** command so that the records are in the same order.

```
$ rwsort --fields=1-10,12-15,20-29 data.rw > /tmp/sorted-data.rw
$ rwsort --fields=1-10,12-15,20-29 other-data.rw   \
  | rwcompare /tmp/sorted-data.rw -
/tmp/sorted-data.rw - differ: record 103363
```

ENVIRONMENT

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwcombine** may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **rwcombine** may use this environment variable. See the FILES section for details.

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

SEE ALSO

rwfileinfo(1), **rwcat(1)**, **rwsort(1)**, **silk(7)**

rwcount

Print traffic summary across time

SYNOPSIS

```
rwcount [--bin-size=SIZE] [--load-scheme=LOADSCHEME]
      [--start-time=START_TIME] [--end-time=END_TIME]
      [--skip-zeroes] [--bin-slots] [--epoch-slots]
      [--timestamp-format=FORMAT] [--no-titles]
      [--no-columns] [--column-separator=CHAR]
      [--no-final-delimiter] [{--delimited | --delimited=CHAR}]
      [--print-filenames] [--copy-input=PATH] [--output-path=PATH]
      [--pager=PAGER_PROG] [--site-config-file=FILENAME]
      [{--legacy-timestamps | --legacy-timestamps={1,0}}]
      {[--xargs] | [--xargs=FILENAME] | [FILE [FILE ...]]}


rwcount --help


rwcount --version
```

DESCRIPTION

rwcount summarizes SiLK flow records across time. It counts the records in the input stream, and groups their byte and packet totals into time bins. **rwcount** produces textual output with one row for each bin.

rwcount reads SiLK Flow records from the files named on the command line or from the standard input when no file names are specified and **--xargs** is not present. To read the standard input in addition to the named files, use `-` or `stdin` as a file name. If an input file name ends in `.gz`, the file is uncompressed as it is read. When the **--xargs** switch is provided, **rwcount** reads the names of the files to process from the named text file or from the standard input if no file name argument is provided to the switch. The input to **--xargs** must contain one file name per line.

rwcount splits each flow record into bins whose size is determined by the argument to the **--bin-size** switch. When that switch is not provided, **rwcount** uses 30-second bins by default.

By default, the first row of data **rwcount** prints is the bin containing the starting time of the earliest record that appears in the input. **rwcount** then prints a row for every bin until it reaches the bin containing the most recent ending time. Rows whose counts are zero are printed unless the **--skip-zero** switch is specified.

The **--start-time** and **--end-time** switches tell **rwcount** to use a specific time for the first row and the final row. The **--start-time** switch always sets the time stamp on the first bin to the specified time. With the **--end-time** switch, **rwcount** computes a maximum end-time by setting any unspecified hour, minute, second, and millisecond field to its maximum value, and the final bin is that which contains the maximum end-time.

When **--start-time** and **--end-time** are both specified, **rwcount** reserves the memory for the bins before it begins processing the records. If the memory cannot be allocated, **rwcount** exits. If this happens, try reducing the time span or increasing the bin-size.

Load Scheme

A router or other flow generator summarizes the traffic it sees into records. In addition to the five-tuple (source port and address, destination port and address, and protocol), the record has its start time, end time, total byte count, and total packet count. There is no way to know how the bytes and packets were distributed during the duration of the record: their distribution could be front-loaded, back-loaded, uniform, et cetera.

When the start and end times of a individual flow record put that record into a single bin, **rwcount** can simply add that record's volume (byte and packet counts) to the bin.

When the duration of a flow record causes it to span multiple bins, **rwcount** must to told how to allocate the volume among the bins. The **--load-scheme** switch determines this, and it has supports the following allocation schemes:

time-proportional

Each bin a flow spans is allocated a percentage of the flow's volume proportional to the amount of the flow's active time that spans the bin. Specifically, **rwcount** divides the total volume of the flow by the duration of the flow, and multiplies the quotient by the time spent in the bin. This models a flow where the volume/second ratio is uniform throughout the flow.

bin-uniform

Each bin a flow spans is allocated an equal portion of the flow's volume. **rwcount** divides the volume of the flow by the number of bins the flow spans, and adds the quotient to each of the bins. In this scheme, the volume/bin ratio is uniform.

start-spike

The bin that contains the flow's start time is allocated all of the flow's volume regardless of the flow's duration. **rwcount** adds the total volume for the flow into the bin containing the start time of the flow. This models a flow that is front-loaded to the point where the entire volume is a single spike occurring in the initial millisecond of flow.

middle-spike

The bin that contains the midpoint between the flow's start time and end time is allocated all of the flow's volume regardless of the flow's duration.

end-spike

The bin that contains the flow's end time is allocated all of the flow's volume regardless of the flow's duration. This models a flow that is back-loaded to the point where the entire volume is a single spike occurring in final millisecond of the flow.

maximum-volume

Each bin the flow spans is allocated *all* of the flow's volume. **rwcount** adds the entire volume for the flow into *every* bin that contains any part of the flow. In theory, the distribution of the bytes in the record could be a spike that occurs at any point during the flow's duration. This scheme allows one to determine, in aggregate, the maximum possible volume that could have occurred during this bin. In this scheme, the `Records` column gives the number of records that were active during the bin.

minimum-volume

For a record that spans multiple bins, each bin is allocated *none* of the flow's volume. That is, **rwcount** acts as though the volume for the flow occurred in some other bin. Since it is possible that a record that spans multiple bins did not contribute any volume to the current bin, this scheme allows one to

determine, in aggregate, the minimum possible volume that may have occurred during this bin. The `Records` column in this scheme, as in the `maximum-volume` scheme, gives the number of flow records that were active during the bin.

Be aware that the "spike" load-schemes allocate the entire flow to a single bin. This can create the impression that there is more traffic occurring during a particular time window that the physical network supports.

The `maximum-volume` and `minimum-volume` schemes are used to compute the maximum and minimum volumes that could have been transferred during any one bin. `maximum-volume` intentionally over-counts the flow volume and `minimum-volume` intentionally under-counts.

To see the effect of the various load-schemes, suppose **rwcount** is using 60-second bins and the input contains two records. The first record begins at 12:03:50, ends at 12:06:20, and contains 9,000 bytes (60 bytes/second for 150 seconds). This record may contribute to bins at 12:03, 12:04, 12:05, and 12:06. The second record begins at 12:04:05 and lasts 15 seconds; this record's volume always contributes its 200 bytes to the 12:04 bin. The **--load-scheme** option splits the byte-counts of the records as follows:

BIN	12:03:00	12:04:00	12:05:00	12:06:00
time-proportional	600	3800	3600	1200
bin-uniform	2250	2450	2250	2250
start-spike	9000	200	0	0
middle-spike	0	200	9000	0
end-spike	0	200	0	9000
maximum-volume	9000	9200	9000	9000
minimum-volume	0	200	0	0

For the record that spans multiple bins: the `time-proportional` scheme assumes 60 bytes/second, the `bin-uniform` scheme divides the volume evenly by the four bins, the `middle-spike` scheme assumes all the volume occurs at 12:05:05, the `maximum-volume` scheme adds the volume to every bin, and the `minimum-volume` scheme ignores the record.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--bin-size=*SIZE*

Denote the size of each time bin, in seconds; defaults to 30 seconds. **rwcount** supports millisecond size bins; *SIZE* may be a floating point value equal to or greater than than 0.001.

--load-scheme=*LOADSCHEME*

Specify how a flow record that spans multiple bins allocates its bytes and packets among the bins. The default scheme is `time-proportional`, which assumes the volume/second ratio of the flow record is constant. See the Load Scheme section for additional information on the load-scheme choices. The *LOADSCHEME* may be one of the following names or numbers; names may be abbreviated to the shortest prefix that is unique.

time-proportional,4

Allocate the volume in proportion to the amount of time the flow spent in the bin.

bin-uniform,0
: Allocate the volume evenly across the bins that contain any part of the flow's duration.

start-spike,1
: Allocate the entire volume to the bin containing the start time of the flow.

middle-spike,3
: Allocate the entire volume to the bin containing the time at the midpoint of the flow.

end-spike,2
: Allocate the entire volume to the bin containing the end time of the flow.

maximum-volume,5
: Allocate the entire volume to *all* of the bins containing any part of the flow.

minimum-volume,6
: Allocate the flow's volume to a bin only if the flow is completely contained within the bin; otherwise ignore the flow.

--start-time=*START_TIME*

Set the time of the first bin to *START_TIME*. When this switch is not given, the first bin is one that holds the starting time of the earliest record. The *START_TIME* may be specified in a format of `yyyy/mm/dd[:HH[:MM[:SS[.sss]]]]` (or `T` may be used in place of `:` to separate the day and hour). The time must be specified to at least day precision, and unspecified hour, minute, second, and millisecond values are set to zero. Whether the date strings represent times in UTC or the local timezone depend on how SiLK was compiled, which can be determined from the `Timezone support` setting in the output from **rwcount --version**. Alternatively, the time may be specified as seconds since the UNIX epoch, and an unspecified milliseconds value is set to 0.

--end-time=*END_TIME*

Set the time of the final bin to *END_TIME*. When this switch is not given, the final bin is one that holds the ending time of the latest record. The format of *END_TIME* is the same as that for *START_TIME*. Unspecified hour, minute, second, and millisecond values are set to 23, 59, 59, and 999 respectively. When *END_TIME* is specified as seconds since the UNIX epoch, an unspecified milliseconds value is set to 999. When both **--start-time** and **--end-time** are used, the *END_TIME* is adjusted so that the final bin represents a complete interval.

--skip-zeroes

Disable printing of bins with no traffic. By default, all bins are printed.

--bin-slots

Use the internal bin index as the label for each bin in the output; the default is to label each bin with the time in a human-readable format.

--epoch-slots

Use the UNIX epoch time (number of seconds since midnight UTC on 1970-01-01) as the label for each bin in the output; the default is to label each bin with the time in a human-readable format. This switch is equivalent to **--timestamp-format=epoch**. This switch is deprecated as of SiLK 3.11.0, and it will be removed in the SiLK 4.0 release.

--timestamp-format=*FORMAT*

Specify the format and/or timezone to use when printing timestamps. When this switch is not specified, the SILK_TIMESTAMP_FORMAT environment variable is checked for a default format and/or timezone. If it is empty or contains invalid values, timestamps are printed in the default format, and the timezone is UTC unless SiLK was compiled with local timezone support. *FORMAT* is a comma-separated list of a format and/or a timezone. The format is one of:

default
Print the timestamps as *YYYY*/*MM*/*DD*T*hh*:*mm*:*ss*.

iso
Print the timestamps as *YYYY*-*MM*-*DD* *hh*:*mm*:*ss*.

m/d/y
Print the timestamps as *MM*/*DD*/*YYYY* *hh*:*mm*:*ss*.

epoch
Print the timestamps as the number of seconds since 00:00:00 UTC on 1970-01-01.

When a timezone is specified, it is used regardless of the default timezone support compiled into SiLK. The timezone is one of:

utc
Use Coordinated Universal Time to print timestamps.

local
Use the TZ environment variable or the local timezone.

--no-titles
Turn off column titles. By default, titles are printed.

--no-columns
Disable fixed-width columnar output.

--column-separator=*C*
Use specified character between columns and after the final column. When this switch is not specified, the default of '|' is used.

--no-final-delimiter
Do not print the column separator after the final column. Normally a delimiter is printed.

--delimited

--delimited=*C*
Run as if **--no-columns --no-final-delimiter --column-sep**=*C* had been specified. That is, disable fixed-width columnar output; if character *C* is provided, it is used as the delimiter between columns instead of the default '|'.

--print-filenames
Print to the standard error the names of input files as they are opened.

--copy-input=*PATH*
Copy all binary SiLK Flow records read as input to the specified file or named pipe. *PATH* may be `stdout` or `-` to write flows to the standard output as long as the **--output-path** switch is specified to redirect **rwcount**'s textual output to a different location.

--output-path=*PATH*
Write the textual output to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output (and bypass the paging program). If *PATH* names an existing file, **rwcount** exits with an error unless the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. If this switch is not given, the output is either sent to the pager or written to the standard output.

--pager=*PAGER_PROG*

When output is to a terminal, invoke the program *PAGER_PROG* to view the output one screen full at a time. This switch overrides the SILK_PAGER environment variable, which in turn overrides the PAGER variable. If the **--output-path** switch is given or if the value of the pager is determined to be the empty string, no paging is performed and all output is written to the terminal.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwcount** searches for the site configuration file in the locations specified in the FILES section.

--legacy-timestamps

--legacy-timestamps=*NUM*

When *NUM* is not specified or is 1, this switch is equivalent to **--timestamp-format=m/d/y**. Otherwise, the switch has no effect. This switch is deprecated as of SiLK 3.0.0, and it will be removed in the SiLK 4.0 release.

--xargs

--xargs=*FILENAME*

Read the names of the input files from *FILENAME* or from the standard input if *FILENAME* is not provided. The input is expected to have one filename per line. **rwcount** opens each named file in turn and reads records from it as if the filenames had been listed on the command line.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

--start-epoch=*START_TIME*

Alias the **--start-time** switch. This switch is deprecated as of SiLK 3.8.0.

--end-epoch=*START_TIME*

Alias the **--end-time** switch. This switch is deprecated as of SiLK 3.8.0.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

To count all web traffic on Feb 12, 2009, into 1 hour bins:

```
$ rwfilter --pass=stdout --start-date=2009/02/12:00        \
       --end-date=2009/02/12:23 --proto=6 --aport=80       \
  | rwcount --bin-size=3600
               Date|      Records|          Bytes|      Packets|
2009/02/12T00:00:00|      1490.49|   578270918.16|    463951.55|
2009/02/12T01:00:00|      1459.33|   596455716.52|    457487.80|
2009/02/12T02:00:00|      1529.06|   562602842.44|    451456.41|
2009/02/12T03:00:00|      1503.89|   562683116.38|    455554.81|
2009/02/12T04:00:00|      1561.89|   590554569.78|    489273.81|
....
```

To bin the records according to their start times, use the **--load-scheme** switch:

```
$ rwfilter ... --pass=stdout       \
  | rwcount --bin-size=3600 --load-scheme=1
               Date|      Records|          Bytes|      Packets|
2009/02/12T00:00:00|      1494.00|   580350969.00|    464952.00|
2009/02/12T01:00:00|      1462.00|   596145212.00|    457871.00|
2009/02/12T02:00:00|      1526.00|   561629416.00|    451088.00|
2009/02/12T03:00:00|      1502.00|   563500618.00|    455262.00|
2009/02/12T04:00:00|      1562.00|   589265818.00|    489279.00|
...
```

To bin the records by their end times: $ rwfilter ... --pass=stdout \ | rwcount --bin-size=3600 --load-scheme=2 Date| Records| Bytes| Packets| 2009/02/12T00:00:00| 1488.00| 577132372.00| 463393.00| 2009/02/12T01:00:00| 1458.00| 596956697.00| 457376.00| 2009/02/12T02:00:00| 1530.00| 562806395.00| 451551.00| 2009/02/12T03:00:00| 1506.00| 562101791.00| 455671.00| 2009/02/12T04:00:00| 1562.00| 591408602.00| 489371.00| ...

To force the hourly bins to run from 30 minutes past the hour, use the **--start-time** switch:

```
$ rwfilter ... --pass=stdout       \
  | rwcount --bin-size=3600 --start-time=2002/12/31:23:30
               Date|      Records|          Bytes|      Packets|
2009/02/12T00:30:00|      1483.26|   581251364.04|    456554.40|
2009/02/12T01:30:00|      1494.00|   575037453.00|    449280.00|
2009/02/12T02:30:00|      1486.36|   559700466.61|    447700.15|
2009/02/12T03:30:00|      1555.23|   588882400.58|    480724.48|
2009/02/12T04:30:00|      1537.79|   564756248.52|    472003.45|
...
```

ENVIRONMENT

SILK_TIMESTAMP_FORMAT

This environment variable is used as the value for **--timestamp-format** when that switch is not provided. *Since SiLK 3.11.0.*

SILK_PAGER

When set to a non-empty string, **rwcount** automatically invokes this program to display its output a screen at a time. If set to an empty string, **rwcount** does not automatically page its output.

PAGER

When set and SILK_PAGER is not set, **rwcount** automatically invokes this program to display its output a screen at a time.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwcount** may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **rwcount** may use this environment variable. See the FILES section for details.

TZ

When the argument to the **--timestamp-format** switch includes `local` or when a SiLK installation is built to use the local timezone, the value of the TZ environment variable determines the timezone in which **rwcount** displays timestamps. (If both of those are false, the TZ environment variable is ignored.) If the TZ environment variable is not set, the machine's default timezone is used. Setting TZ to the empty string or 0 causes timestamps to be displayed in UTC. For system information on the TZ variable, see **tzset(3)** or **environ(7)**. (To determine if SiLK was built with support for the local timezone, check the `Timezone support` value in the output of **rwcount --version**.) The TZ environment variable is also used when **rwcount** parses the timestamp specified in the **--start-time** or **--end-time** switches if SiLK is built with local timezone support.

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

SEE ALSO

rwfilter(1), **rwuniq(1)**, **silk(7)**, **tzset(3)**, **environ(7)**

BUGS

Unlike **rwuniq(1)**, **rwcount** does not support counting the number of distinct IPs in a bin. However, using the **--bin-time** switch on **rwuniq** can provide time-based binning similar to what **rwcount** supports. Note that **rwuniq** always bins by the each record's start-time (similar to **rwcount --load-factor=1**), and there is no support in **rwuniq** for dividing a SiLK record among multiple time bins.

rwcut

Print selected fields of binary SiLK Flow records

SYNOPSIS

```
rwcut [{--fields=FIELDS | --all-fields}]
      {[--start-rec-num=START_NUM] [--end-rec-num=END_NUM]
       | [--tail-recs=TAIL_START_NUM]}
      [--num-recs=REC_COUNT] [--dry-run] [--icmp-type-and-code]
      [--timestamp-format=FORMAT] [--epoch-time]
      [--ip-format=FORMAT] [--integer-ips] [--zero-pad-ips]
      [--integer-sensors] [--integer-tcp-flags]
      [--no-titles] [--no-columns] [--column-separator=CHAR]
      [--no-final-delimiter] [{--delimited | --delimited=CHAR}]
      [--print-filenames] [--copy-input=PATH] [--output-path=PATH]
      [--pager=PAGER_PROG] [--site-config-file=FILENAME]
      [--ipv6-policy={ignore,asv4,mix,force,only}]
      [{--legacy-timestamps | --legacy-timestamps={1,0}}]
      [--plugin=PLUGIN [--plugin=PLUGIN ...]]
      [--python-file=PATH [--python-file=PATH ...]]
      [--pmap-file=MAPNAME:PATH [--pmap-file=MAPNAME:PATH ...]]
      [--pmap-column-width=NUM]
      {[--xargs] | [--xargs=FILENAME] | [FILE [FILE ...]]}

rwcut [--pmap-file=MAPNAME:PATH [--pmap-file=MAPNAME:PATH ...]]
      [--plugin=PLUGIN ...] [--python-file=PATH ...] --help

rwcut [--pmap-file=MAPNAME:PATH [--pmap-file=MAPNAME:PATH ...]]
      [--plugin=PLUGIN ...] [--python-file=PATH ...] --help-fields

rwcut --version
```

DESCRIPTION

rwcut reads binary SiLK Flow records and prints the user-selected record attributes (or fields) to the terminal in a textual, bar-delimited (|) format. See the EXAMPLES section below for sample output.

rwcut reads SiLK Flow records from the files named on the command line or from the standard input when no file names are specified and **--xargs** is not present. To read the standard input in addition to the named files, use `-` or `stdin` as a file name. If an input file name ends in `.gz`, the file is uncompressed as it is read. When the **--xargs** switch is provided, **rwcut** reads the names of the files to process from the named text file or from the standard input if no file name argument is provided to the switch. The input to **--xargs** must contain one file name per line.

The user may provide the **--fields** switch to select the record attributes to print. When **--fields** is not specified **rwcut** prints the source and destination IP address, source and destination port, protocol, packet count, byte count, TCP flags, start time, duration, end time, and the sensor name. The fields are printed in the order in which they occur in the **--fields** switch. Fields may be repeated.

A subset of the input records may be selected by using the **--start-rec-num**, **--end-rec-num**, **--num-recs**, and **--tail-recs** switches.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--fields=*FIELDS*

FIELDS contains the list of flow attributes (a.k.a. fields or columns) to print. The columns will be displayed in the order the fields are specified. Fields may be repeated. *FIELDS* is a comma separated list of field-names, field-integers, and ranges of field-integers; a range is specified by separating the start and end of the range with a hyphen (**-**). Field-names are case-insensitive. Example:

```
--fields=stime,10,1-5
```

If the **--fields** switch is not given, *FIELDS* defaults to:

```
sIP,dIP,sPort,dPort,protocol,packets,bytes,flags,sTime,dur,eTime,sensor
```

The complete list of built-in fields that the SiLK tool suite supports follows, though note that not all fields are present in all SiLK file formats; when a field is not present, its value is 0.

sIP,1
: source IP address

dIP,2
: destination IP address

sPort,3
: source port for TCP and UDP, or equivalent

dPort,4
: destination port for TCP and UDP, or equivalent

protocol,5
: IP protocol

packets,pkts,6
: packet count

bytes,7
: byte count

flags,8
: bit-wise OR of TCP flags over all packets

sTime,9
: starting time of flow in millisecond resolution

duration,10
: duration of flow in millisecond resolution

eTime,11
: end time of flow in millisecond resolution

sensor,12
: name or ID of sensor at the collection point

class,20
: class of sensor at the collection point

type,21
: type of sensor at the collection point

sTime+msec,22
: starting time of flow including milliseconds (milliseconds are always displayed); this field is deprecated as of SiLK 3.8.1, and it will be removed in the SiLK 4.0 release

eTime+msec,23
: end time of flow including milliseconds (milliseconds are always displayed); this field is deprecated as of SiLK 3.8.1, and it will be removed in the SiLK 4.0 release

dur+msec,24
: duration of flow including milliseconds (milliseconds are always displayed); this field is deprecated as of SiLK 3.8.1, and it will be removed in the SiLK 4.0 release

iType
: the ICMP type value for ICMP or ICMPv6 flows and empty for non-ICMP flows. This field was introduced in SiLK 3.8.1.

iCode
: the ICMP code value for ICMP or ICMPv6 flows and empty for non-ICMP flows. See note at `iType`.

icmpTypeCode,25
: equivalent to `iType`,`iCode`. This field is deprecated as of SiLK 3.8.1.

Many SiLK file formats do not store the following fields and their values will always be 0; they are listed here for completeness:

in,13
: router SNMP input interface or vlanId if packing tools were configured to capture it (see **sensor.conf(5)**)

out,14
: router SNMP output interface or postVlanId

nhIP,15
: router next hop IP

Enhanced flow metering software (such as **yaf(1)**) may provide flow information elements in addition to those found in NetFlow. SiLK stores some of these elements in the fields named below. For flows without this additional information, the field's value is always 0.

initialFlags,26
: TCP flags on first packet in the flow

sessionFlags,27
: bit-wise OR of TCP flags on the second through final packets in the flow

attributes,28
: flow attributes set by the flow generator:

  `S`
  : all the packets in this flow record are exactly the same size

`F`
: flow generator saw additional packets in this flow following a packet with a FIN flag (excluding ACK packets)

`T`
: flow generator prematurely created a record for a long-running connection due to a timeout. (When the flow generator **yaf(1)** is run with the **--silk** switch, it will prematurely create a flow and mark it with `T` if the byte count of the flow cannot be stored in a 32-bit value.)

`C`
: flow generator created this flow as a continuation of long-running connection, where the previous flow for this connection met a timeout (or a byte threshold in the case of **yaf**).

Consider a long-running ssh session that exceeds the flow generator's *active* timeout. (This is the active timeout since the flow generator creates a flow for a connection that still has activity). The flow generator will create multiple flow records for this ssh session, each spanning some portion of the total session. The first flow record will be marked with a `T` indicating that it hit the timeout. The second through next-to-last records will be marked with `TC` indicating that this flow both timed out and is a continuation of a flow that timed out. The final flow will be marked with a `C`, indicating that it was created as a continuation of an active flow.

application,29
: guess as to the content of the flow. Some software that generates flow records from packet data, such as **yaf**, will inspect the contents of the packets that make up a flow and use traffic signatures to label the content of the flow. SiLK calls this label the *application*; **yaf** refers to it as the *appLabel*. The application is the port number that is traditionally used for that type of traffic (see the */etc/services* file on most UNIX systems). For example, traffic that the flow generator recognizes as FTP will have a value of 21, even if that traffic is being routed through the standard HTTP/web port (80).

The following fields provide a way to label the IPs or ports on a record. These fields require external files to provide the mapping from the IP or port to the label:

sType,16
: for the source IP address, the value 0 if the address is non-routable, 1 if it is internal, or 2 if it is routable and external. Uses the mapping file specified by the SILK_ADDRESS_TYPES environment variable, or the *address_types.pmap* mapping file, as described in **addrtype(3)**.

dType,17
: as **sType** for the destination IP address

scc,18
: for the source IP address, a two-letter country code abbreviation denoting the country where that IP address is located. Uses the mapping file specified by the SILK_COUNTRY_CODES environment variable, or the *country_codes.pmap* mapping file, as described in **ccfilter(3)**. The abbreviations are those defined by ISO 3166-1 (see for example https://www.iso.org/iso-3166-country-codes.html or https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2) or the following special codes: **--** N/A (e.g. private and experimental reserved addresses); **a1** anonymous proxy; **a2** satellite provider; **o1** other

dcc,19
: as **scc** for the destination IP

src-*map-name*
: label contained in the prefix map file associated with *map-name*. If the prefix map is for IP addresses, the label is that associated with the source IP address. If the prefix map is for protocol/port pairs, the label is that associated with the protocol and source port. See also the description of the **--pmap-file** switch below and the **pmapfilter(3)** manual page.

dst-*map-name*

as **src-*map-name*** for the destination IP address or the protocol and destination port.

sval

as **src-*map-name*** when no map-name is associated with the prefix map file

dval

as **dst-*map-name*** when no map-name is associated with the prefix map file

Finally, the list of built-in fields may be augmented by the run-time loading of PySiLK code or plug-ins written in C (also called shared object files or dynamic libraries), as described by the **--python-file** and **--plugin** switches.

--all-fields

Instruct **rwcut** to print all known fields. This switch may not be combined with the **--fields** switch. This switch suppresses error messages from the plug-ins.

--plugin=*PLUGIN*

Augment the list of fields by using run-time loading of the plug-in (shared object) whose path is *PLUGIN*. The switch may be repeated to load multiple plug-ins. The creation of plug-ins is described in the **silk-plugin(3)** manual page. When *PLUGIN* does not contain a slash (/), **rwcut** will attempt to find a file named *PLUGIN* in the directories listed in the FILES section. If **rwcut** finds the file, it uses that path. If *PLUGIN* contains a slash or if **rwcut** does not find the file, **rwcut** relies on your operating system's **dlopen(3)** call to find the file. When the SILK_PLUGIN_DEBUG environment variable is non-empty, **rwcut** prints status messages to the standard error as it attempts to find and open each of its plug-ins.

--start-rec-num=*START_NUM*

Begin printing with the *START_NUM*'th record by skipping the first *START_NUM*-1 records. The default is 1; that is, to start printing at the first record; *START_NUM* must be a positive integer. If *START_NUM* is greater than the number of input records, **rwcut** only outputs the title. This switch may not be combined with the **--tail-recs** switch. When using multiple input files, records are treated as a single stream for the purposes of the **--start-rec-num**, **--end-rec-num**, **--tail-recs**, and **--num-recs** switches. This switch does not affect the records written to the stream specified by **--copy-input**.

--end-rec-num=*END_NUM*

Stop printing after the *END_NUM*'th record. When *END_NUM* is 0, the default, printing stops once all input records have been printed; that is, *END_NUM* is effectively infinity. If this value is non-zero, it must not be less than *START_NUM*. This switch may not be combined with the **--tail-recs** switch. When using multiple input files, records are treated as a single stream for the purposes of the **--start-rec-num**, **--end-rec-num**, **--tail-recs**, and **--num-recs** switches. This switch does not affect the records written to the stream specified by **--copy-input**.

--tail-recs=*TAIL_START_NUM*

Begin printing once **rwcut** is *TAIL_START_NUM* records from end of the input stream, where *TAIL_START_NUM* is a positive integer. **rwcut** will print the remaining records in the input stream unless **--num-recs** is also specified and is less than *TAIL_START_NUM*. The **--tail-recs** switch is similar to the **--start-rec-num** switch except it counts from the end of the input stream. This switch may not be combined with the **--start-rec-num** and **--end-rec-num** switches. When using multiple input files, records are treated as a single stream for the purposes of the **--start-rec-num**, **--end-rec-num**, **--tail-recs**, and **--num-recs** switches. This switch does not affect the records written to the stream specified by **--copy-input**.

--num-recs=*REC_COUNT*

Print no more than *REC_COUNT* records. Specifying a *REC_COUNT* of 0 will print all records, which is the default. This switch is ignored under the following conditions: When both **--start-rec-num** and **--end-rec-num** are specified; when only **--end-rec-num** is given and *END_NUM* is less than *REC_COUNT*; when **--tail-recs** is specified and *TAIL_START_NUM* is less than *REC_COUNT*. When using multiple input files, records are treated as a single stream for the purposes of the **--start-rec-num**, **--end-rec-num**, **--tail-recs**, and **--num-recs** switches. This switch does not affect the records written to the stream specified by **--copy-input**.

--dry-run

Causes **rwcut** to print the column headers and exit. Useful for testing.

--icmp-type-and-code

Unlike TCP or UDP, ICMP messages do not use ports, but instead have types and codes. Specifying this switch will cause **rwcut** to print, for ICMP records, the message's type and code in the sPort and dPort columns, respectively. Use of this switch has been discouraged since SiLK 0.9.10. As for SiLK 3.8.1, this switch is deprecated and it will be removed in SiLK 4.0; use the **iType** and **iCode** fields instead.

--timestamp-format=*FORMAT*

Specify the format, timezone, and/or modifier to use when printing timestamps. When this switch is not specified, the SILK_TIMESTAMP_FORMAT environment variable is checked for a format, timezone, and modifier. If it is empty or contains invalid values, timestamps are printed in the default format, and the timezone is UTC unless SiLK was compiled with local timezone support. *FORMAT* is a comma-separated list of a format, a timezone, and/or a modifier. The format is one of:

default

Print the timestamps as *YYYY/MM/DD*T*hh*:*mm*:*ss.sss*.

iso

Print the timestamps as *YYYY-MM-DD hh*:*mm*:*ss.sss*.

m/d/y

Print the timestamps as *MM/DD/YYYY hh*:*mm*:*ss.sss*.

epoch

Print the timestamps as the number of seconds since 00:00:00 UTC on 1970-01-01.

When a timezone is specified, it is used regardless of the default timezone support compiled into SiLK. The timezone is one of:

utc

Use Coordinated Universal Time to print timestamps.

local

Use the TZ environment variable or the local timezone.

One modifier is available:

no-msec

Truncate the milliseconds value on the timestamps and on the duration field. When milliseconds are truncated, the sum of the printed start time and duration may not equal the printed end time.

--epoch-time

Print timestamps as epoch time (number of seconds since midnight GMT on 1970-01-01). This switch is equivalent to **--timestamp-format=epoch**, it is deprecated as of SiLK 3.0.0, and it will be removed in the SiLK 4.0 release.

--ip-format=*FORMAT*

Specify how IP addresses are printed, where *FORMAT* is a comma-separated list of the arguments described below. When this switch is not specified, the SILK_IP_FORMAT environment variable is checked for a value and that format is used if it is valid. The default *FORMAT* is `canonical` according to whether the individual flow record is marked as IPv4 or IPv6. *Since SiLK 3.7.0.*

canonical

Print IP addresses in the canonical format. For an IPv4 record, use dot-separated decimal (`192.0.2.1`). For an IPv6 record, use either colon-separated hexadecimal (`2001:db8::1`) or a mixed IPv4-IPv6 representation for IPv4-mapped IPv6 addresses (the ::ffff:0:0/96 netblock, e.g., `::ffff:192.0.2.1`) and IPv4-compatible IPv6 addresses (the ::/96 netblock other than ::/127, e.g., `::192.0.2.1`).

no-mixed

Print IP addresses in the canonical format (`192.0.2.1` or `2001:db8::1`) but do not used the mixed IPv4-IPv6 representations. For example, use `::ffff:c000:201` instead of `::ffff:192.0.2.1`. *Since SiLK 3.17.0.*

decimal

Print IP addresses as integers in decimal format. For example, print `192.0.2.1` and `2001:db8::1` as `3221225985` and `42540766411282592856903984951653826561`, respectively.

hexadecimal

Print IP addresses as integers in hexadecimal format. For example, print `192.0.2.1` and `2001:db8::1` as `c00000201` and `20010db800000000000000000000001`, respectively.

zero-padded

Make all IP address strings contain the same number of characters by padding numbers with leading zeros. For example, print `192.0.2.1` and `2001:db8::1` as `192.000.002.001` and `2001:0db8:0000:0000:0000:0000:0000:0001`, respectively. For IPv6 addresses, this setting implies `no-mixed`, so that `::ffff:192.0.2.1` is printed as `0000:0000:0000:0000:0000:ffff:c000:0201`. As of SiLK 3.17.0, may be combined with any of the above, including `decimal` and `hexadecimal`.

The following arguments modify certain IP addresses prior to printing. These arguments may be combined with the above formats.

map-v4

Change IPv4 addresses to IPv4-mapped IPv6 addresses (addresses in the ::ffff:0:0/96 netblock) prior to formatting. *Since SiLK 3.17.0.*

unmap-v6

Change any IPv4-mapped IPv6 addresses (addresses in the ::ffff:0:0/96 netblock) to IPv4 addresses prior to formatting. *Since SiLK 3.17.0.*

The following argument is also available:

force-ipv6

Set *FORMAT* to `map-v4`,`no-mixed`.

--integer-ips

Print IP addresses as integers. This switch is equivalent to **--ip-format=decimal**, it is deprecated as of SiLK 3.7.0, and it will be removed in the SiLK 4.0 release.

--zero-pad-ips

Print IP addresses as fully-expanded, zero-padded values in their canonical form. This switch is equivalent to **--ip-format=zero-padded**, it is deprecated as of SiLK 3.7.0, and it will be removed in the SiLK 4.0 release.

--integer-sensors

Print the integer ID of the sensor rather than its name.

--integer-tcp-flags

Print the TCP flag fields (flags, initialFlags, sessionFlags) as an integer value. Typically, the characters `F,S,R,P,A,U,E,C` are used to represent the TCP flags.

--no-titles

Turn off column titles. By default, titles are printed.

--no-columns

Disable fixed-width columnar output.

--column-separator=*C*

Use specified character between columns and after the final column. When this switch is not specified, the default of '|' is used.

--no-final-delimiter

Do not print the column separator after the final column. Normally a delimiter is printed.

--delimited

--delimited=*C*

Run as if **--no-columns --no-final-delimiter --column-sep**=*C* had been specified. That is, disable fixed-width columnar output; if character *C* is provided, it is used as the delimiter between columns instead of the default '|'.

--print-filenames

Print to the standard error the names of input files as they are opened.

--copy-input=*PATH*

Copy all binary SiLK Flow records read as input to the specified file or named pipe. *PATH* may be `stdout` or `-` to write flows to the standard output as long as the **--output-path** switch is specified to redirect **rwcut**'s textual output to a different location.

--output-path=*PATH*

Write the textual output to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output (and bypass the paging program). If *PATH* names an existing file, **rwcut** exits with an error unless the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. If this switch is not given, the output is either sent to the pager or written to the standard output.

--pager=*PAGER_PROG*

When output is to a terminal, invoke the program *PAGER_PROG* to view the output one screen full at a time. This switch overrides the SILK_PAGER environment variable, which in turn overrides the PAGER variable. If the **--output-path** switch is given or if the value of the pager is determined to be the empty string, no paging is performed and all output is written to the terminal.

--ipv6-policy=*POLICY*

Determine how IPv4 and IPv6 flows are handled when SiLK has been compiled with IPv6 support. When the switch is not provided, the SILK_IPV6_POLICY environment variable is checked for a policy. If it is also unset or contains an invalid policy, the *POLICY* is **mix**. When SiLK has not been compiled with IPv6 support, IPv6 flows are always ignored, regardless of the value passed to this switch or in the SILK_IPV6_POLICY variable. The supported values for *POLICY* are:

ignore

Ignore any flow record marked as IPv6, regardless of the IP addresses it contains. Only records marked as IPv4 will be printed.

asv4

Convert IPv6 flow records that contain addresses in the ::ffff:0:0/96 netblock (that is, IPv4-mapped IPv6 addresses) to IPv4 and ignore all other IPv6 flow records.

mix

Process the input as a mixture of IPv4 and IPv6 flow records.

force

Convert IPv4 flow records to IPv6, mapping the IPv4 addresses into the ::ffff:0:0/96 netblock.

only

Print only flow records that are marked as IPv6 and ignore IPv4 flow records in the input.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwcut** searches for the site configuration file in the locations specified in the FILES section.

--legacy-timestamps

--legacy-timestamps=*NUM*

When *NUM* is not specified or is 1, this switch is equivalent to **--timestamp-format=m/d/y,no-msec**. Otherwise, the switch has no effect. This switch is deprecated as of SiLK 3.0.0, and it will be removed in the SiLK 4.0 release.

--xargs

--xargs=*FILENAME*

Read the names of the input files from *FILENAME* or from the standard input if *FILENAME* is not provided. The input is expected to have one filename per line. **rwcut** opens each named file in turn and reads records from it as if the filenames had been listed on the command line.

--help

Print the available options and exit. Specifying switches that add new fields or additional switches before **--help** will allow the output to include descriptions of those fields or switches.

--help-fields

Print the description and alias(es) of each field and exit. Specifying switches that add new fields before **--help-fields** will allow the output to include descriptions of those fields.

--version

Print the version number and information about how SiLK was configured, then exit the application.

--pmap-file=*PATH*

--pmap-file=*MAPNAME*:*PATH*

Load the prefix map file located at *PATH* and create fields named src-*map-name* and dst-*map-name* where *map-name* is either the *MAPNAME* part of the argument or the map-name specified when the file was created (see **rwpmapbuild(1)**). If no map-name is available, **rwcut** names the fields `sval` and `dval`. Specify *PATH* as `-` or `stdin` to read from the standard input. The switch may be repeated to load multiple prefix map files, but each prefix map must use a unique map-name. The **--pmap-file** switch(es) must precede the **--fields** switch. See also **pmapfilter(3)**.

--pmap-column-width=*NUM*

When printing a label associated with a prefix map, this switch gives the maximum number of characters to use when displaying the textual value of the field.

--python-file=*PATH*

When the SiLK Python plug-in is used, **rwcut** reads the Python code from the file *PATH* to define additional fields for possible output. This file should call **register_field()** for each field it wishes to define. For details and examples, see the **silkpython(3)** and **pysilk(3)** manual pages.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

The standard output from **rwcut** resembles the following (with the text wrapped for readability):

```
            sIP|            dIP|sPort|dPort|pro|\
   10.30.30.31|   10.70.70.71|   80|36761|  6|\

       packets|     bytes|    flags|\
             7|      3227|FS PA    |\

                   sTime| duration|                  eTime|senso|
 2003/01/01T00:00:14.625|    3.959|2003/01/01T00:00:18.584|EDGE1|
```

The first line of the output is the title line which shows the names of the selected fields; the **--no-titles** switch will disable the printing of the title line. The second line and onward will contain the printed representation of the records, with one line per record.

A common use of **rwcut** is to read the output of **rwfilter(1)**. For example, to see representative TCP traffic:

```
$ rwfilter --start-date=2002/01/19:00 --end-date=2002/01/19:01     \
      --proto=6 --pass=stdout                                      \
  | rwcut
```

To see only selected fields, use the **--fields** switch. For example, to print only the protocol for each record in the input file *data.rw*, use:

```
$ rwcut --fields=proto  data.rw
```

The **silkpython(3)** manual page provides examples that use PySiLK to create and print arbitrary fields for **rwcut**.

The order of the *FIELDS* is significant, and fields can be repeated. For example, here is a case where in addition to the default fields of 1-12, you also to prefix each row with an integer form of the destination IP and the start time to make processing by another tool (e.g., a spreadsheet) easier. However, within the default fields of 1-12, you want to see dotted-decimal IP addresses. (The **num2dot(1)** tool converts the numeric fields in *column positions* three and four to dotted quad IPs.)

```
$ rwfilter ... --pass=stdout \
  | rwcut --fields=2,9,1-12 --ip-format=decimal --timestamp-format=epoch \
  | num2dot --ip-field=3,4
```

Both of the following commands print the title line and the first record in the input stream:

```
$ rwcut --num-recs=1  data.rw
```

```
$ rwcut --end-rec-num=1  data.rw
```

The following prints all records except the first (plus the title):

```
$ rwcut --start-rec-num=2  data.rw
```

These three commands print only the second record:

```
$ rwcut --no-title --start-rec-num=2 --num-recs=1  data.rw
```

```
$ rwcut --no-title --start-rec-num=2 --end-rec-num=2  data.rw
```

```
$ rwcut --no-title --end-rec-num=2 --num-recs=1  data.rw
```

This command prints the title line and the final record in the input stream:

```
$ rwcut --tail-recs=1  data.rw
```

This command prints the next to last record in the input stream:

```
$ rwcut --no-title --tail-recs=2 --num-recs=1  data.rw
```

Using the `sIP` and `dIP` fields can be confusing when the file you are examining contains both incoming and outgoing flow records. To make the output more clear, consider using the **int-ext-fields(3)** plug-in. The plug-in defines four additional fields representing the external IP address, the external port, the internal IP address, and the internal port. The plug-in requires the user to specify which class/type pairs are incoming and which are outgoing. See its manual page for additional information.

```
$ rwcut --fields=sip,sport,dip,dport,proto,type \
       --num-rec=8 data.rw
            sIP|sPort|            dIP|dPort|pro|   type|
192.168.111.201|29617|   172.24.2.123|   53| 17|    out|
   172.24.2.123|   53|192.168.111.201|29617| 17|     in|
192.168.111.201|29618|  10.252.217.50|   22|  6|    out|
  10.252.217.50|   22|192.168.111.201|29618|  6|     in|
192.168.204.193|   68|    172.30.2.67|   67| 17|    out|
    172.30.2.67|   67|192.168.204.193|   68| 17|     in|
  10.239.85.193|29897|192.168.228.153|   25|  6|     in|
192.168.228.153|   25|  10.239.85.193|29897|  6|    out|

$ export INCOMING_FLOWTYPES=all/in,all/inweb
$ export OUTGOING_FLOWTYPES=all/out,all/outweb
$ rwcut --plugin=int-ext-fields.so                          \
       --fields=int-ip,int-port,ext-ip,ext-port,proto,type \
       --num-rec=8 data.rw
         int-ip|int-p|         ext-ip|ext-p|pro|   type|
192.168.111.201|29617|   172.24.2.123|   53| 17|    out|
192.168.111.201|29617|   172.24.2.123|   53| 17|     in|
192.168.111.201|29618|  10.252.217.50|   22|  6|    out|
192.168.111.201|29618|  10.252.217.50|   22|  6|     in|
192.168.204.193|   68|    172.30.2.67|   67| 17|    out|
192.168.204.193|   68|    172.30.2.67|   67| 17|     in|
192.168.228.153|   25|  10.239.85.193|29897|  6|     in|
192.168.228.153|   25|  10.239.85.193|29897|  6|    out|
```

ENVIRONMENT

SILK_IPV6_POLICY

This environment variable is used as the value for **--ipv6-policy** when that switch is not provided.

SILK_IP_FORMAT

This environment variable is used as the value for **--ip-format** when that switch is not provided. *Since SiLK 3.11.0.*

SILK_TIMESTAMP_FORMAT

This environment variable is used as the value for **--timestamp-format** when that switch is not provided. *Since SiLK 3.11.0.*

SILK_PAGER

When set to a non-empty string, **rwcut** automatically invokes this program to display its output a screen at a time. If set to an empty string, **rwcut** does not automatically page its output.

PAGER

When set and SILK_PAGER is not set, **rwcut** automatically invokes this program to display its output a screen at a time.

PYTHONPATH

This environment variable is used by Python to locate modules. When **--python-file** is specified, **rwcut** must load the Python files that comprise the PySiLK package, such as *silk/__init__.py*. If this

silk/ directory is located outside Python's normal search path (for example, in the SiLK installation tree), it may be necessary to set or modify the PYTHONPATH environment variable to include the parent directory of *silk/* so that Python can find the PySiLK module.

SILK_PYTHON_TRACEBACK

When set, Python plug-ins will output traceback information on Python errors to the standard error.

SILK_COUNTRY_CODES

This environment variable allows the user to specify the country code mapping file that **rwcut** uses when computing the scc and dcc fields. The value may be a complete path or a file relative to the SILK_PATH. See the FILES section for standard locations of this file.

SILK_ADDRESS_TYPES

This environment variable allows the user to specify the address type mapping file that **rwcut** uses when computing the sType and dType fields. The value may be a complete path or a file relative to the SILK_PATH. See the FILES section for standard locations of this file.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwcut** may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files and plug-ins, **rwcut** may use this environment variable. See the FILES section for details.

TZ

When the argument to the **--timestamp-format** switch includes `local` or when a SiLK installation is built to use the local timezone, the value of the TZ environment variable determines the timezone in which **rwcut** displays timestamps. (If both of those are false, the TZ environment variable is ignored.) If the TZ environment variable is not set, the machine's default timezone is used. Setting TZ to the empty string or 0 causes timestamps to be displayed in UTC. For system information on the TZ variable, see **tzset(3)** or **environ(7)**. (To determine if SiLK was built with support for the local timezone, check the `Timezone support` value in the output of **rwcut --version**.)

SILK_PLUGIN_DEBUG

When set to 1, **rwcut** prints status messages to the standard error as it attempts to find and open each of its plug-ins. In addition, when an attempt to register a field fails, **rwcut** prints a message specifying the additional function(s) that must be defined to register the field in **rwcut**. Be aware that the output can be rather verbose.

FILES

$SILK_ADDRESS_TYPES

$SILK_PATH/share/silk/address_types.pmap

$SILK_PATH/share/address_types.pmap

/usr/local/share/silk/address_types.pmap

/usr/local/share/address_types.pmap

Possible locations for the address types mapping file required by the sType and dType fields.

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

$SILK_COUNTRY_CODES

$SILK_PATH/share/silk/country_codes.pmap

$SILK_PATH/share/country_codes.pmap

/usr/local/share/silk/country_codes.pmap

/usr/local/share/country_codes.pmap

Possible locations for the country code mapping file required by the scc and dcc fields.

${SILK_PATH}/lib64/silk/

${SILK_PATH}/lib64/

${SILK_PATH}/lib/silk/

${SILK_PATH}/lib/

/usr/local/lib64/silk/

/usr/local/lib64/

/usr/local/lib/silk/

/usr/local/lib/

Directories that **rwcut** checks when attempting to load a plug-in.

NOTES

If you are interested in only a few fields, use the **--fields** option to reduce the volume of data to be produced. For example, if you are checking to see which internal host got hit with the slammer worm (signature: UDP, destPort 1434, pkt size 404), then the following **rwfilter**, **rwcut** combination will be much faster than simply using default values:

```
$ rwfilter --proto-17 --dport=1434 --bytes-per-packet=404-404      \
  | rwcut --fields=dip,stime
```

SEE ALSO

rwfilter(1), **num2dot(1)**, **rwpmapbuild(1)**, **addrtype(3)**, **ccfilter(3)**, **int-ext-fields(3)**, **pmapfilter(3)**, **silk-plugin(3)**, **silkpython(3)**, **pysilk(3)**, **sensor.conf(5)**, **silk(7)**, **yaf(1)**, **dlopen(3)**, **tzset(3)**, **environ(7)**

rwdedupe

Eliminate duplicate SiLK Flow records

SYNOPSIS

```
rwdedupe [--ignore-fields=FIELDS] [--packets-delta=NUM]
      [--bytes-delta=NUM] [--stime-delta=NUM] [--duration-delta=NUM]
      [--temp-directory=DIR_PATH] [--buffer-size=SIZE]
      [--note-add=TEXT] [--note-file-add=FILE]
      [--compression-method=COMP_METHOD] [--print-filenames]
      [--output-path=PATH] [--site-config-file=FILENAME]
      {[--xargs] | [--xargs=FILENAME] | [FILE [FILE ...]]}

rwdedupe --help

rwdedupe --help-fields

rwdedupe --version
```

DESCRIPTION

rwdedupe reads SiLK Flow records from one or more input sources. Records that appear in the input file(s) multiple times will only appear in the output stream once; that is, duplicate records are not written to the output. The SiLK Flows are written to the file specified by the **--output-path** switch or to the standard output when the **--output-path** switch is not provided and the standard output is not connected to a terminal.

Note: As part of its processing, **rwdedupe** re-orders the records before writing them.

rwdedupe reads SiLK Flow records from the files named on the command line or from the standard input when no file names are specified and **--xargs** is not present. To read the standard input in addition to the named files, use `-` or `stdin` as a file name. If an input file name ends in `.gz`, the file is uncompressed as it is read. When the **--xargs** switch is provided, **rwdedupe** reads the names of the files to process from the named text file or from the standard input if no file name argument is provided to the switch. The input to **--xargs** must contain one file name per line.

By default, **rwdedupe** will consider one record to be a duplicate of another when *all* the fields in the records match exactly. From another point on view, any difference in two records results in both records appearing in the output. Note that *all* means every field that exists on a SiLK Flow record. The complete list of fields is specified in the description of **--ignore-fields** in the OPTIONS section below.

To have **rwdedupe** ignore fields in the comparison, specify those fields in the **--ignore-fields** switch. When **--ignore-fields**=*FIELDS* is specified, a record is considered a duplicate of another if all fields *except* those in *FIELDS* match exactly. **rwdedupe** will treat *FIELDS* as being identical across all records. Put another way, if the only difference between two records is in the *FIELDS* fields, only one of those records will be written to the output.

The **--packets-delta**, **--bytes-delta**, **--stime-delta** and **--duration-delta** switches allow for "fuzziness" in the input. For example, if **--stime-delta**=*NUM* is specified and the only difference between two records is in the sTime fields, and the fields are within *NUM* milliseconds of each other, only one record will be written to the output.

During its processing, **rwdedupe** will try to allocate a large (near 2GB) in-memory array to hold the records. (You may use the **--buffer-size** switch to change this maximum buffer size.) If more records are read than will fit into memory, the in-core records are temporarily stored on disk as described by the **--temp-directory** switch. When all records have been read, the on-disk files are merged to produce the output.

By default, the temporary files are stored in the */tmp* directory. Because of the sizes of the temporary files, it is strongly recommended that */tmp not* be used as the temporary directory, and **rwdedupe** will print a warning when */tmp* is used. To modify the temporary directory used by **rwdedupe**, provide the **--temp-directory** switch, set the SILK_TMPDIR environment variable, or set the TMPDIR environment variable.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--ignore-fields=*FIELDS*

Ignore the fields listed in *FIELDS* when determining if two flow records are identical; that is, treat *FIELDS* as being identical across all flows. By default, all fields are treated as significant.

FIELDS is a comma separated list of field-names, field-integers, and ranges of field-integers; a range is specified by separating the start and end of the range with a hyphen (**-**). Field-names are case-insensitive. Example:

```
--ignore-fields=stime,12-15
```

The list of supported fields are:

sIP,1
: source IP address

dIP,2
: destination IP address

sPort,3
: source port for TCP and UDP, or equivalent

dPort,4
: destination port for TCP and UDP, or equivalent

protocol,5
: IP protocol

packets,pkts,6
: packet count

bytes,7
: byte count

flags,8
: bit-wise OR of TCP flags over all packets

sTime,9
: starting time of flow (milliseconds resolution)

duration,10
duration of flow (milliseconds resolution)

sensor,12
name or ID of sensor at the collection point

in,13
router SNMP input interface or vlanId if packing tools were configured to capture it (see **sensor.conf(5)**)

out,14
router SNMP output interface or postVlanId

nhIP,15
router next hop IP

class,20,type,21
class and type of sensor at the collection point (represented internally by a single value)

initialFlags,26
TCP flags on first packet in the flow

sessionFlags,27
bit-wise OR of TCP flags over all packets except the first in the flow

attributes,28
flow attributes set by flow generator

application,29
guess as to the content of the flow. Some software that generates flow records from packet data, such as **yaf(1)**, will inspect the contents of the packets that make up a flow and use traffic signatures to label the content of the flow. SiLK calls this label the *application*; **yaf** refers to it as the *appLabel*. The application is the port number that is traditionally used for that type of traffic (see the */etc/services* file on most UNIX systems). For example, traffic that the flow generator recognizes as FTP will have a value of 21, even if that traffic is being routed through the standard HTTP/web port (80).

--packets-delta=*NUM*
Treat the packets field on two records as being the same if the values differ by *NUM* packets or less. If not specified, the default is 0.

--bytes-delta=*NUM*
Treat the bytes field on two records as being the same if the values differ by *NUM* bytes or less. If not specified, the default is 0.

--stime-delta=*NUM*
Treat the start-time field on two records as being the same if the values differ by *NUM* milliseconds or less. If not specified, the default is 0.

--duration-delta=*NUM*
Treat the duration field on two records as being the same if the values differ by *NUM* milliseconds or less. If not specified, the default is 0.

--temp-directory=*DIR_PATH*
Specify the name of the directory in which to store data files temporarily when more records have been read that will fit into RAM. This switch overrides the directory specified in the SILK_TMPDIR environment variable, which overrides the directory specified in the TMPDIR variable, which overrides the default, */tmp*.

--buffer-size=*SIZE*

Set the maximum size of the buffer to use for holding the records, in bytes. A larger buffer means fewer temporary files need to be created, reducing the I/O wait times. The default maximum for this buffer is near 2GB. The *SIZE* may be given as an ordinary integer, or as a real number followed by a suffix `K`, `M` or `G`, which represents the numerical value multiplied by 1,024 (kilo), 1,048,576 (mega), and 1,073,741,824 (giga), respectively. For example, 1.5K represents 1,536 bytes, or one and one-half kilobytes. (This value does **not** represent the absolute maximum amount of RAM that **rwdedupe** will allocate, since additional buffers will be allocated for reading the input and writing the output.)

--output-path=*PATH*

Write the binary SiLK Flow records to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output. If *PATH* names an existing file, **rwdedupe** exits with an error unless the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. If this switch is not given, the output is written to the standard output. Attempting to write the binary output to a terminal causes **rwdedupe** to exit with an error.

--note-add=*TEXT*

Add the specified *TEXT* to the header of the output file as an annotation. This switch may be repeated to add multiple annotations to a file. To view the annotations, use the **rwfileinfo(1)** tool.

--note-file-add=*FILENAME*

Open *FILENAME* and add the contents of that file to the header of the output file as an annotation. This switch may be repeated to add multiple annotations. Currently the application makes no effort to ensure that *FILENAME* contains text; be careful that you do not attempt to add a SiLK data file as an annotation.

--compression-method=*COMP_METHOD*

Specify the compression library to use when writing output files. If this switch is not given, the value in the SILK_COMPRESSION_METHOD environment variable is used if the value names an available compression method. When no compression method is specified, output to the standard output or to named pipes is not compressed, and output to files is compressed using the default chosen when SiLK was compiled. The valid values for *COMP_METHOD* are determined by which external libraries were found when SiLK was compiled. To see the available compression methods and the default method, use the **--help** or **--version** switch. SiLK can support the following *COMP_METHOD* values when the required libraries are available.

none

Do not compress the output using an external library.

zlib

Use the **zlib(3)** library for compressing the output, and always compress the output regardless of the destination. Using zlib produces the smallest output files at the cost of speed.

lzo1x

Use the *lzo1x* algorithm from the LZO real time compression library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead.

snappy

Use the *snappy* library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead. *Since SiLK 3.13.0.*

best

Use lzo1x if available, otherwise use snappy if available, otherwise use zlib if available. Only compress the output when writing to a file.

--print-filenames

Print to the standard error the names of input files as they are opened.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwdedupe** searches for the site configuration file in the locations specified in the FILES section.

--xargs

--xargs=*FILENAME*

Read the names of the input files from *FILENAME* or from the standard input if *FILENAME* is not provided. The input is expected to have one filename per line. **rwdedupe** opens each named file in turn and reads records from it as if the filenames had been listed on the command line.

--help

Print the available options and exit.

--help-fields

Print the description and alias(es) of each field and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

LIMITATIONS

When the temporary files and the final output are stored on the same file volume, **rwdedupe** will require approximately twice as much free disk space as the size of input data.

When the temporary files and the final output are on different volumes, **rwdedupe** will require between 1 and 1.5 times as much free space on the temporary volume as the size of the input data.

EXAMPLE

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line.

Suppose you have made several **rwfilter(1)** runs to find interesting traffic:

```
$ rwfilter --start-date=2008/02/04 ... --pass=data1.rw
$ rwfilter --start-date=2008/02/04 ... --pass=data2.rw
$ rwfilter --start-date=2008/02/04 ... --pass=data3.rw
$ rwfilter --start-date=2008/02/04 ... --pass=data4.rw
```

You now want to merge that traffic into a single output file, but you want to ensure that any records appearing in multiple output files are only counted once. You can use **rwdedupe** to merge the output files to a single file, *data.rw*:

```
$ rwdedupe data1.rw data2.rw data3.rw data4.rw --output=data.rw
```

ENVIRONMENT

SILK_TMPDIR

When set and **--temp-directory** is not specified, **rwdedupe** writes the temporary files it creates to this directory. SILK_TMPDIR overrides the value of TMPDIR.

TMPDIR

When set and SILK_TMPDIR is not set, **rwdedupe** writes the temporary files it creates to this directory.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_COMPRESSION_METHOD

This environment variable is used as the value for **--compression-method** when that switch is not provided. *Since SiLK 3.13.0.*

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwdedupe** may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **rwdedupe** may use this environment variable. See the FILES section for details.

SILK_TEMPFILE_DEBUG

When set to 1, **rwdedupe** prints debugging messages to the standard error as it creates, re-opens, and removes temporary files.

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

${SILK_TMPDIR}/

${TMPDIR}/

/tmp/

Directory in which to create temporary files.

SEE ALSO

rwfilter(1), **rwfileinfo(1)**, **sensor.conf(5)**, **silk(7)**, **yaf(1)**, **zlib(3)**

rwfglob

Print files that **rwfilter**'s File Selection switches will access

SYNOPSIS

```
rwfglob { [--class=CLASS] [--type={all | TYPE[,TYPE ...]}]
          | [--flowtypes=CLASS/TYPE[,CLASS/TYPE ...]] }
      [--sensors=SENSOR[,SENSOR ...]]
      [--start-date=YYYY/MM/DD[:HH] [--end-date=YYYY/MM/DD[:HH]]]
      [--data-rootdir=ROOT_DIRECTORY] [--site-config-file=FILENAME]
      [--print-missing-files] [--no-block-check] [--no-file-names]
      [--no-summary]

rwfglob [--data-rootdir=ROOT_DIRECTORY]
      [--site-config-file=FILENAME] --help

rwfglob --version
```

DESCRIPTION

rwfglob accepts the same File Selection Switches of **rwfilter(1)** and prints, to the standard output, the pathnames of the files that **rwfilter** would process, one file name per line. At the end, a summary is printed to the standard output of the number of files that **rwfglob** found. To suppress the printing of the file names and/or the summary, specify the **--no-file-names** and/or **--no-summary** switches, respectively.

By default, **rwfglob** only prints the names of files that exist. When the **--print-missing-files** switch is provided, **rwfglob** prints, to the standard error, the names of files that it did not find, one file name per line, preceded by the text 'Missing '. To redirect the output of **--print-missing-files** to the standard output, use the following in a Bourne-compatible shell:

```
$ rwfglob --print-missing-files ... 2>&1
```

Read Selection Argument Values from a File

As of SiLK 3.20, the Selection Switches **--class**, **--type**, **--flowtypes**, and **--sensors** accept a value in the form "@*PATH*", where `@` is the "at" character (ASCII 0x40) and *PATH* names a file or a path to a file. For example, the following reads the name of types from the file *t.txt* and uses the sensors `S3`, `S7`, and the names and/or IDs read from */tmp/sensor.txt*:

```
rwfglob --type=@t.txt --sensors=S3,@/tmp/sensor.txt,S7
```

Multiple @*PATH* values are allowed within a single argument. If the name of the file is `-`, the names are read from the standard input.

The file must be a text file. Blank lines are ignored as are comments, which begin with the `#` character and continue to the end of the line. Whitespace at the beginning and end of a line is ignored as is whitespace that surrounds commas; all other whitespace within a line is significant.

A file may contain a value on each line and/or multiple values on a line separated by commas and optional whitespace. For example:

```
# Sensor 4
      S4
# The first sensors
S0, S1,S2
S3     # Sensor 3
```

An attempt to use an @*PATH* directive in a file is an error.

When **rwfglob** is parsing the name of a file, it converts the sequences `@,` and `@@` to `,` and `@`, respectively. For example, **--class=@cl@@ss.txt@,v** reads the class from the file *cl@ss.txt,v*. It is an error if any other character follows an embedded `@` (**--flowtypes=@f@il** contains `@i`) or if a single `@` occurs at the end of the name (**--sensor=@errat@**).

Offline Storage Support

For each file it finds, **rwfglob** will check the size of the file and the number of blocks allocated to the file. If the block count is zero but the file size is non-zero, **rwfglob** treats the file as existing but as residing *on tape*. The names of these files are printed to the standard output, but each name is preceded by the text ' \t*** ON_TAPE ***' where '\t' represents a tab character. The summary line will include the number of files that **rwfglob** believes are on tape. To suppress this check and to remove the count from the summary line, use the **--no-block-check** switch.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

Selection Switches

This set of switches are the same as those used by **rwfilter** to select the files to process. At least one of these switches must be provided.

--class={*CLASS* | @*PATH*}

The **--class** switch is used to specify a group of files to print. Only a single class may be selected with the **--class** switch; for multiple classes, use the **--flowtypes** switch. The argument may be "@*PATH*" which causes **rwfglob** to open the file *PATH* and read the class name from it; see Read Selection Argument Values from a File for details. Classes are defined in the **silk.conf(5)** site configuration file. If neither the **--class** nor **--flowtypes** option is given, the default-class as specified in *silk.conf* is used. To see the available classes and the default class, either examine the output from **rwfglob --help** or invoke **rwsiteinfo(1)** with the switch **--fields=class,default-class**.

--type={`all` | *TYPE*[,*TYPE*,@*PATH* ...]}

The **--type** predicate further specifies data within the selected *CLASS* by listing the *TYPE*s of traffic to process. The switch takes either the keyword `all` to select all types for *CLASS* or a comma-separated list of type names and "@*PATH*" directives, where @*PATH* tells **rwfglob** to read type names from the file *PATH*; see Read Selection Argument Values from a File for details. Types are defined in *silk.conf*, they typically refer to the direction of the flow, and they may vary by class. When neither the **--type** nor **--flowtypes** switch is given, a list of default types is used: The default-type list is determined by

the value of *CLASS*, and the default types often include only incoming traffic. To see the available types and the default types for each class, examine the **--help** output of **rwfglob** or run **rwsiteinfo** with **--fields=class,type,default-type**.

--flowtypes=*CLASS/TYPE*[,*CLASS/TYPE*,*@PATH* ...]

The **--flowtypes** predicate provides an alternate way to specify class/type pairs. The **--flowtypes** switch allows a single **rwfglob** invocation to print filenames from multiple classes. The keyword `all` may be used for the *CLASS* and/or *TYPE* to select all classes and/or types. As of SiLK 3.20.0, the arguments may also include "@*PATH*" which causes **rwfglob** to open the file *PATH* and read the class/type pairs from it; see Read Selection Argument Values from a File.

--sensors=*SENSOR*[,*SENSOR*,*SENSOR-GROUP*,*@PATH* ...]

The **--sensors** switch is used to select data from specific sensors. The parameter is a comma separated list of sensor names, sensor IDs (integers), ranges of sensor IDs, sensor group names, and/or "@*PATH*" directives. As described in Read Selection Argument Values from a File, @*PATH* tells **rwfglob** to read the names of the sensors from the file *PATH*. Sensors and sensor groups are defined in the **silk.conf(5)** site configuration file, and the **rwsiteinfo(1)** command can be used to print a mapping of sensor names to IDs and classes (**--fields=sensor,id-sensor,class:list**). When the **--sensors** switch is not specified, the default is to use all sensors which are valid for the specified class(es). Support for using sensor group names was added in SiLK 3.21.0.

--start-date=*YYYY/MM/DD[:HH]*

--end-date=*YYYY/MM/DD[:HH]*

The date predicates indicate which days and hours to consider when creating the list of files. The dates may be expressed as seconds since the UNIX epoch or in `YYYY/MM/DD[:HH]` format, where the hour is optional. A `T` may be used in place of the `:` to separate the day and hour. Whether the `YYYY/MM/DD[:HH]` strings represent times in UTC or the local timezone depend on how SiLK was compiled. To determine how your version of SiLK was compiled, see the `Timezone support` setting in the output from **rwfglob --version**.

When times are expressed in `YYYY/MM/DD[:HH]` format:

- When both **--start-date** and **--end-date** are specified to hour precision, all hours within that time range are processed.
- When **--start-date** is specified to day precision, the hour specified in **--end-date** (if any) is ignored, and files for all dates between midnight on **start-date** and 23:59 on **end-date** are processed.
- When **--start-date** is specified to hour precision and **--end-date** is specified to day precision, the hour of the start-date is used as the hour for the end-date.
- When **--end-date** is not specified and **--start-date** is specified to day precision, files for that complete day are processed.
- When **--end-date** is not specified and **--start-date** is specified to hour precision, files for that single hour are processed.

When at least one time is expressed as seconds since the UNIX epoch:

- When **--end-date** is specified in epoch seconds, the given **--start-date** and **--end-date** are considered to be in hour precision.
- When **--start-date** is specified in epoch seconds and **--end-date** is specified in `YYYY/MM/DD[:HH]` format, the start-date is considered to be in day precision if it divisible by 86400, and hour precision otherwise.

- When **--start-date** is specified in epoch seconds and **--end-date** is not given, the start-date is considered to be in hour-precision.

When neither **--start-date** nor **--end-date** is given, **rwfglob** prints all files for the current day.

It is an error to specify **--end-date** without specifying **--start-date**.

--data-rootdir=*ROOT_DIRECTORY*

Tell **rwfglob** to use *ROOT_DIRECTORY* as the root of the data repository, which overrides the location given in the SILK_DATA_ROOTDIR environment variable, which in turn overrides the location that was compiled into **rwfglob** (/data).

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwfglob** searches for the site configuration file in the locations specified in the FILES section.

--print-missing-files

This option prints to the standard error the names of the files that **rwfglob** expected to find but did not. The file names are preceded by the text 'Missing '; each file name appears on a separate line. This switch is useful for debugging, but the list of files it produces can be misleading. For example, suppose there is a decommissioned sensor that still appears in the *silk.conf* file; **rwfglob** considers these data files as *missing* even though their absence is expected. Use the output from this switch judiciously.

Application Switches

--no-block-check

This option instructs **rwfglob** not to check whether the file exists *on tape* by checking whether the number of blocks allocated to the file is zero. By default, **rwfglob** precedes a file name that has a block count of 0 with the text ' \t*** ON_TAPE ***'.

--no-file-names

This option instructs **rwfglob** not to print the names of the files that it successfully finds. By default, **rwfglob** prints the names of the files it finds and a summary line showing the number of files it found. When both this switch and **--print-missing-files** are specified, **rwfglob** prints only the names of missing files (and the summary).

--no-summary

This option instructs **rwfglob** not to print the summary line (that is, the line that shows the number of files found). By default, **rwfglob** prints the names of the files it finds and a summary line showing the number of files it found.

--help

Print the available options and exit. The available classes and types will be included in output; you may specify a different root directory or site configuration file before **--help** to see the classes and types available for that site.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line.

Looking at a day on a single sensor:

```
$ rwfglob --start=2003/10/11 --sensor=2
/data/in/2003/10/11/in-GAMMA_20031011.23
/data/in/2003/10/11/in-GAMMA_20031011.22
/data/in/2003/10/11/in-GAMMA_20031011.21
/data/in/2003/10/11/in-GAMMA_20031011.20
/data/in/2003/10/11/in-GAMMA_20031011.19
/data/in/2003/10/11/in-GAMMA_20031011.18
/data/in/2003/10/11/in-GAMMA_20031011.17
/data/in/2003/10/11/in-GAMMA_20031011.16
/data/in/2003/10/11/in-GAMMA_20031011.15
/data/in/2003/10/11/in-GAMMA_20031011.14
/data/in/2003/10/11/in-GAMMA_20031011.13
/data/in/2003/10/11/in-GAMMA_20031011.12
/data/in/2003/10/11/in-GAMMA_20031011.11
/data/in/2003/10/11/in-GAMMA_20031011.10
/data/in/2003/10/11/in-GAMMA_20031011.09
/data/in/2003/10/11/in-GAMMA_20031011.08
/data/in/2003/10/11/in-GAMMA_20031011.07
/data/in/2003/10/11/in-GAMMA_20031011.06
/data/in/2003/10/11/in-GAMMA_20031011.05
/data/in/2003/10/11/in-GAMMA_20031011.04
/data/in/2003/10/11/in-GAMMA_20031011.03
/data/in/2003/10/11/in-GAMMA_20031011.02
/data/in/2003/10/11/in-GAMMA_20031011.01
/data/in/2003/10/11/in-GAMMA_20031011.00
globbed 24 files; 0 on tape
```

If you only want the summary, specify **--no-file-names**

```
$ rwfglob --start-date=2003/10/11 --sensor=2 --no-file-names
globbed 24 files; 0 on tape
```

ENVIRONMENT

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. This value overrides the compiled-in value, and **rwfglob** uses it unless the **--data-rootdir** switch is specified. In addition, **rwfglob** may use this value when searching for the SiLK site configuration file. See the FILES section for details.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **rwfglob** may use this environment variable. See the FILES section for details.

TZ

When a SiLK installation is built to use the local timezone (to determine if this is the case, check the `Timezone support` value in the output from **rwfglob --version**), the value of the TZ environment variable determines the timezone in which **rwfglob** parses timestamps. (The date on the filenames that **rwfglob** returns are always in UTC.) If the TZ environment variable is not set, the default timezone is used. Setting TZ to 0 or the empty string causes timestamps to be parsed as UTC. The value of the TZ environment variable is ignored when the SiLK installation uses utc. For system information on the TZ variable, see **tzset(3)** or **environ(7)**.

FILES

${SILK_CONFIG_FILE}

ROOT_DIRECTORY/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided, where *ROOT_DIRECTORY/* is the directory **rwfglob** is using as the root of the data repository.

${SILK_DATA_ROOTDIR}/

/data/

Locations for the root directory of the data repository when the **--data-rootdir** switch is not specified.

SEE ALSO

rwfilter(1), **rwsiteinfo(1)**, **silk.conf(5)**, **silk(7)**, **tzset(3)**, **environ(7)**

NOTES

The ability to use @*PATH* in **--class**, **--type**, **--flowtypes**, and **--sensors** was added in SiLK 3.20.0.

As of SiLK 3.20.0, **--types** is an alias for **--type**.

The **--sensors** switch also accepts the names of groups defined in the **silk.conf(5)** file as of SiLK 3.21.0.

The output of **--print-missing-files** goes to the standard error, while all other output goes to the standard output. To redirect the output of **--print-missing-files** to the standard output, use the following in a Bourne-compatible shell:

```
$ rwfglob --print-missing-files ... 2>&1
```

The **--print-missing-files** option needs to be smarter about what files are really missing.

The block count check is of unknown portability across different tape-farm systems.

rwfileinfo

Print information about a SiLK file

SYNOPSIS

```
rwfileinfo [--fields=FIELDS] [--summary] [--no-titles]
      [--site-config-file=FILENAME]
      {--xargs | --xargs=FILENAME | FILE [FILE...]}

rwfileinfo --help

rwfileinfo --help-fields

rwfileinfo --version
```

DESCRIPTION

rwfileinfo prints information about a binary SiLK file that can be determined by reading the file's header and by moving quickly over the data blocks in the file.

rwfileinfo requires one or more filename arguments to be given on the command line or the use of the **--xargs** switch. When the **--xargs** switch is provided, **rwfileinfo** reads the names of the files to process from the named text file or from the standard input if no file name argument is provided to the switch. The input to **--xargs** must contain one file name per line. **rwfileinfo** does not read a SiLK file's content from the standard input by default, but it does when either `-` or `stdin` is given as a filename argument.

When the **--summary** switch is given, **rwfileinfo** first prints the information for each individual file and then prints the number of files processed, the sum of the individual file sizes, and the sum of the individual record counts.

Field Descriptions

By default, **rwfileinfo** prints the following information for each file argument. Use the **--fields** switch to modify which pieces of information are printed.

(**rwfileinfo** prints each field in the order in which support for that field was added to SiLK. The field descriptions are presented here in a more logical order.)

file-size

The size of the file on disk as reported by the operating system. **rwfileinfo** prints 0 for the file-size when reading from the standard input.

version

Every binary file written by SiLK has a version number field. Since SiLK 1.0.0, the version number field has been used to indicate the general structure (or layout) of the file. The file structure adopted in SiLK 1.0.0 uses a version number of 16 and has a *header section* and a *data section*. The header section begins with 16 bytes that specify well-defined values, and those bytes are followed by one or more variably-sized *header entries*. The specifics of the data section depend on the content of the file.

header-length

The header-length field shows the number of octets required by header (i.e., the initial 16 bytes and the header entries). Since everything after the header is data, the header-length is the starting offset of the data section. The smallest header length is 24 bytes, but typically the header is padded to be an integer multiple of the record-length. The header-length that **rwfileinfo** prints for a file is determined dynamically by reading the file's header.

silk-version

When a SiLK tool creates a binary file, the tool writes the current SiLK release number (such as 3.9.0) into the file's header as a way to help diagnose issues should a bug with a particular release of SiLK be discovered in the future.

byte-order

Every SiLK file has a byte-order or *endian* field. SiLK uses the machine's native representation of integers when writing data, and this field shows what representation the file contains. `BigEndian` is network byte order and `littleEndian` is used by Intel chips. The **rwswapbytes(1)** tool changes a file's integer representation, and some tools have a **--byte-order** switch that allows the user to specify the integer representation of output files. The header-section of a file is always written in network byte order.

compression

SiLK tools may use the zlib library (http://zlib.net/), the LZO library (http://www.oberhumer.com/opensource/lzo/), or the snappy library (http://google.github.io/snappy/) to compress the data section of a file. The compression field specifies which library (if any) was used to compress the data section. If a file is compressed with a library that was not included in an installation of SiLK, SiLK is unable to read the data section of the file. Many SiLK tools accept the **--compression-method** switch to choose a particular compression method. (The compression field does not indicate whether the entire file has been compressed with an external compression utility such as **gzip(1)**.)

format

Every binary file written by SiLK has two fields in the header that specify exactly what the file contains: the format and the record-version. In general, the *format* indicates the content type of the file and the *record-version* indicates the evolution of that content.

The contents of a file whose format is `FT_IPSET`, `FT_RWBAG`, or `FT_PREFIXMAP` is fairly obvious (an IPset, a Bag, a prefix map).

There are many different file formats for writing SiLK Flow records, but the SiLK analysis tools largely use a single Flow file format. That format is `FT_RWIPV6ROUTING` if SiLK has been compiled with IPv6 support, or `FT_RWGENERIC` otherwise. A file that uses the `FT_RWGENERIC` format is only capable of holding IPv4 addresses.

The other SiLK Flow file formats are created by **rwflowpack(8)** as it writes flow records to the repository. These formats often omit fields and use reduced bit-sizes for fields to reduce the space required for an individual flow record.

The record-version field indicates changes within the general type specified by the format field. For example, SiLK incremented the record-version of the formats that hold flow records when the resolution of record timestamps was changed from seconds to milliseconds.

record-version

Together with the **format** fields specifies the contents of the file. See the discussion of **format** for details.

record-length

Files created by SiLK 1.0.0 and later have a record length field. This field contains the length of an individual record, and this value is dependent on the format and record-version fields described above. Some files (such as those containing IPsets or prefix maps) do not write individual records to the output, and the record length is 1 for these files.

count-records

The count-records field is generated dynamically by determining the length the data section would require if it were completely uncompressed and dividing it by the record-length. When the record-length is 1 (such as for IPset files), the count-records field does not provide much information beyond the length of the uncompressed data. For an uncompressed file, adding header-length to the product of count-records and record-length is equal to the file-size.

The fields given above are either present in the well-defined header or are computed by reading the file.

The following fields are generated by reading the header entries and determining if one or more header entries of the specified type are present. The field is not printed in the output when the header entry is not present in the file.

command-lines

Many of the SiLK tools write a header entry to the output file that contains the command line invocation used to create that file, and some of the SiLK tools also copy the command line history from their input files to the output file. (The **--invocation-strip** switch on the tools can be used to prevent copying and recording of the invocation.) The command lines are stored in individual header entries and this field displays those entries with the most recent invocation at the end of the list.

The command line history is has a couple of issues:

- When multiple input files are used to create a single output, the entries are stored as a list, and this makes it is difficult to know which set of command line entries are associated with which input file.
- When a SiLK tool creates multiple output files (e.g., when using both **--pass** and **--fail** to **rwfilter(1)**), the tool writes the same command line entry to each output file. Some context in addition to the command line history may be needed to know which branch of that tool a particular file represents.

annotations

Most of SiLK tools that create binary output files provide the **--note-add** and **--note-file-add** switches which allow an arbitrary annotation to be added to the header of a file. Some tools also copy the annotations from the source files to the destination files. The annotations are stored in individual header entries and this field displays those entries.

ipset

The IPset writing tools (**rwset(1)**, **rwsetbuild(1)**, **rwsettool(1)**, **rwaggbagtool(1)**, and **rwbagtool(1)**) support the following output formats for IPset data structures:

2

May hold only IPv4 addresses and does not have an ipset header entry.

3

May hold IPv4 or IPv6 addresses and is readable by SiLK 3.0 and later. It contains a header entry that describes the IPset data structure, and the entry specifies the number of nodes, the number of branches from each node, the number of leaves, the size of the nodes and leaves, and which node is the root of the tree.

4

May hold IPv4 or IPv6 addresses and is readable by SiLK 3.7 and later. The file's header entry specifies whether the file contains IPv4 addresses or IPv6 addresses.

5

May hold only IPv6 addresses and is readable by SiLK 3.14 and later. The header entry specifies that the file contains IPv6 data.

bag

Since SiLK 3.0.0, the tools that write binary Bag files (**rwbag(1)**, **rwbagbuild(1)**, and **rwbagtool(1)**) have written a header entry that specifies the type and size of the key and of the counter in the file.

aggregate-bag

The tools **rwaggbag(1)**, **rwaggbagbuild(1)**, and **rwaggbagtool(1)** write a header entry that contains the field types that comprise the key and the counter.

prefix-map

When using **rwpmapbuild(1)** to create a prefix map file, a string that specifies a *mapname* may be provided. **rwpmapbuild** writes the mapname to a header entry in the prefix map file. The mapname is used to generate command line switches or field names when the **--pmap-file** switch is specified to several of the SiLK tools (see **pmapfilter(3)** for details). When displaying the mapname, **rwfileinfo** prefixes it with the string `v1:` which denotes a version number for the prefix-map header entry. (The version number is printed for completeness.)

packed-file-info

When **rwflowpack(8)** creates a SiLK Flow file for the repository, all the records in the file have the same starting hour, the same sensor, and the same flowtype (class/type pair). **rwflowpack** writes a header entry to the file that contains these values, and this field displays those values. (To print the names for the sensor and flowtype, the **silk.conf(5)** file must be accessible.)

probe-name

When **flowcap(8)** creates a SiLK flow file, it adds a header entry specifying the name of the probe from which the data was collected.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--fields=*FIELDS*

Specify what information to print for each file argument on the command line. *FIELDS* is a comma separated list of field-names, field-integers, and ranges of field-integers; a range is specified by separating the start and end of the range with a hyphen (**-**). Field-names are case-insensitive and may be shortened to a unique prefix. When the **--fields** option is not given, all fields are printed if the file contains the necessary information. The fields are always printed in the order they appear here regardless of the order they are specified in *FIELDS*.

The possible field values are given next with a brief description of each. For a full description of each field, see Field Descriptions above.

format,1

The contents of the file as a name and the corresponding hexadecimal ID.

version,2

An integer describing the layout or structure of the file.

byte-order,3

Either `BigEndian` or `littleEndian` to indicate the representation used to store integers in the file (network or non-network byte order).

compression,4

The compression library (if any) used to compress the data-section of the file, specified as a name and its decimal ID.

header-length,5

The octet length of the file's header; alternatively the offset where data begins.

record-length,6

The octet length of a single record or the value 1 if the file's content is not record-based.

count-records,7

The number of records in the file, computed by dividing the uncompressed data length by the record-length.

file-size,8

The size of the file on disk as reported by the operating system.

command-lines,9

The command line invocation used to generate this file.

record-version,10

The version of the records contained in the file.

silk-version,11

The release of SiLK that wrote this file.

packed-file-info,12

For a repository Flow file generated by **rwflowpack(8)**, this prints the timestamp of the starting hour, the flowtype, and the sensor of each flow record in the file.

probe,13

For a Flow file generated by **flowcap(8)**, the name of the probe where the flow records where initially collected.

annotations,14

The notes (annotations) that users have added to the file's header.

prefix-map,15

For a prefix map file, the `mapname` that was set when the file was created by **rwpmapbuild(1)**.

ipset,16

For an IPset file whose record-version is 3, a description of the tree data structure. For an IPset file whose record-version is 4, the type of IP addresses (IPv4 or IPv6).

bag,17

For a bag file, the type and size of the key and of the counter.

aggregate-bag,18

For an aggregate bag file, the field types that comprise the key and the counter.

--summary

After the data for each individual file is printed, print a summary that shows the number of files processed, the sum of the individual file sizes, and the total number of records contained in those files.

--no-titles

Suppress printing of the file name and field names. The output contains only the values, where each value is printed left-justified on a single line.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwfileinfo** searches for the site configuration file in the locations specified in the FILES section.

--xargs

--xargs=*FILENAME*

Read the names of the input files from *FILENAME* or from the standard input if *FILENAME* is not provided. The input is expected to have one filename per line. **rwfileinfo** opens each named file in turn and prints its information as if the filenames had been listed on the command line. *Since SiLK 3.15.0.*

--help

Print the available options and exit.

--help-fields

Print a description of each field, its alias, and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLE

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line.

Get information about the file *tcp-data.rw*:

```
$ rwfileinfo tcp-data.rw
tcp-data.rw:
  format(id)          FT_RWGENERIC(0x16)
  version             16
  byte-order          littleEndian
  compression(id)     none(0)
  header-length       208
  record-length       52
  record-version      5
  silk-version        1.0.1
  count-records       7
  file-size           572
  command-lines
                  1  rwfilter --proto=6 --pass=tcp-data.rw ...
  annotations
                  1  This is some interesting TCP data
```

Return a single value which is the number of records in the file *tcp-data.rw*:

```
$ rwfileinfo --no-titles --field=count-records tcp-data.rw
7
```

ENVIRONMENT

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwfileinfo** may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **rwfileinfo** may use this environment variable. See the FILES section for details.

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

SEE ALSO

rwfilter(1), **rwaggbag(1)**, **rwaggbagbuild(1)**, **rwaggbagtool(1)**, **rwbag(1)**, **rwbagbuild(1)**, **rwbagtool(1)**, **rwpmapbuild(1)**, **rwset(1)**, **rwsetbuild(1)**, **rwsettool(1)** **rwswapbytes(1)**, **silk.conf(5)**, **pmapfilter(3)**, **flowcap(8)**, **rwflowpack(8)**, **silk(7)**, **gzip(1)**

rwfilter

Choose which SiLK Flow records to process

SYNOPSIS

```
rwfilter INPUT_ARGS OUTPUT_ARGS PARTITIONING_ARGS [MISC_ARGS]
```

Selection switches, input switches, or input files are required:

```
rwfilter ...
      {{ [--class=CLASS] [--type={all | TYPE[,TYPE ...]}]
         | [--flowtypes=CLASS/TYPE[,CLASS/TYPE ...]] }
       [--sensors=SENSOR[,SENSOR ...]]
       [--start-date=YYYY/MM/DD[:HH] [--end-date=YYYY/MM/DD[:HH]]]
       [--data-rootdir=ROOT_DIRECTORY] [--print-missing-files] }
      | [--input-pipe=INPUT_PATH]
      | [--xargs] | [--xargs=INPUT_PATH]
      | [INPUT_PATH [INPUT_PATH...]]
```

One or more output switches are required:

```
rwfilter ...
      [--all-destination=ALL_PATH [--all-destination=ALL_PATH ...]]
      [--fail-destination=FAIL_PATH [--fail-destination=FAIL_PATH ...]]
      [--pass-destination=PASS_PATH [--pass-destination=PASS_PATH ...]]
      [{ --print-statistics[=STATS_PATH]
         | --print-volume-statistics[=STATS_PATH] }]
```

One or more partitioning switches are required:

```
rwfilter ...
      [--ack-flag=SCALAR] [--active-time=TIME_WINDOW]
      [{--any-address=IP_WILDCARD | --not-any-address=IP_WILDCARD}]
      [--any-cc=COUNTRY_CODE_LIST]
      [{--any-cidr=IP_OR_CIDR_LIST | --not-any-cidr=IP_OR_CIDR_LIST}]
      [--any-index=INTEGER_LIST]
      [{--anyset=IP_SET_FILENAME | --not-anyset=IP_SET_FILENAME}]
      [--aport=INTEGER_LIST] [--application=INTEGER_LIST]
      [--attributes=ATTRIBUTES_LIST]
      [--bytes=INTEGER_RANGE] [--bytes-per-packet=DECIMAL_RANGE]
      [--cwr-flag=SCALAR]
      [{--daddress=IP_WILDCARD | --not-daddress=IP_WILDCARD}]
      [--dcc=COUNTRY_CODE_LIST]
      [{--dcidr=IP_OR_CIDR_LIST | --not-dcidr=IP_OR_CIDR_LIST}]
      [{--dipset=IP_SET_FILENAME | --not-dipset=IP_SET_FILENAME}]
      [--dport=INTEGER_LIST] [--dtype=SCALAR]
      [--duration=DECIMAL_RANGE] [--ece-flag=SCALAR]
      [--etime=TIME_WINDOW] [--fin-flag=SCALAR]
```

```
       [--flags-all=HIGH_MASK_FLAGS_LIST]
       [--flags-initial=HIGH_MASK_FLAGS_LIST]
       [--flags-session=HIGH_MASK_FLAGS_LIST]
       [--icmp-code=INTEGER_LIST] [--icmp-type=INTEGER_LIST]
       [--input-index=INTEGER_LIST] [--ip-version=INTEGER_LIST]
       [--ipa-src-expr=IPA_EXPR] [--ipa-dst-expr=IPA_EXPR]
       [--ipa-any-expr=IPA_EXPR]
       [{--next-hop-id=IP_WILDCARD | --not-next-hop-id=IP_WILDCARD}]
       [{--nhcidr=IP_OR_CIDR_LIST | --not-nhcidr=IP_OR_CIDR_LIST}]
       [{--nhipset=IP_SET_FILENAME | --not-nhipset=IP_SET_FILENAME}]
       [--output-index=INTEGER_LIST] [--packets=INTEGER_RANGE]
       [--pmap-file=MAPNAME:PATH [--pmap-file=MAPNAME:PATH ...]
        { [--pmap-src-MAPNAME=LABELS] [--pmap-dst-MAPNAME=LABELS]
          [--pmap-any-MAPNAME=LABELS] } ]
       [--protocol=INTEGER_LIST] [--psh-flag=SCALAR]
       [--python-expr=PYTHON_EXPR]
       [--python-file=FILENAME [--python-file=FILENAME ...]]
       [--rst-flag=SCALAR]
       [{--saddress=IP_WILDCARD | --not-saddress=IP_WILDCARD}]
       [--scc=COUNTRY_CODE_LIST]
       [{--scidr=IP_OR_CIDR_LIST | --not-scidr=IP_OR_CIDR_LIST}]
       [{--sipset=IP_SET_FILENAME | --not-sipset=IP_SET_FILENAME}]
       [--sport=INTEGER_LIST] [--stime=TIME_WINDOW] [--stype=SCALAR]
       [--syn-flag=SCALAR] [--tcp-flags=TCP_FLAGS]
       [--tuple-file=TUPLE_FILENAME { [--tuple-fields=FIELDS]
                                      [--tuple-direction=DIRECTION]
                                      [--tuple-delimiter=CHAR] } ]
       [--urg-flag=SCALAR]
```

Miscellaneous switches:

```
rwfilter ...
       [--compression-method=COMP_METHOD] [--dry-run]
       [--max-fail-records=N] [--max-pass-records=N]
       [--note-add=TEXT] [--note-file-add=FILE]
       [--plugin=PLUGIN [--plugin=PLUGIN ...]]
       [--print-filenames] [--site-config-file=FILENAME]
       [--threads=N]
```

Help switches:

```
rwfilter [--pmap-file=MAPNAME:PATH [--pmap-file=MAPNAME:PATH ...]]
       [--plugin=PLUGIN ...] [--python-file=PATH]
       [--data-rootdir=ROOT_DIRECTORY] [--site-config-file=FILENAME]
       --help


rwfilter --version
```

DESCRIPTION

rwfilter serves two purposes: (1) It acts as an interface to the data store to select which SiLK Flow records to process, and (2) it partitions those records into one or more *pass* and/or *fail* streams.

The Selection Switches let one choose flow records from the SiLK data store by specifying where the flow was collected (its sensor), the date of collection, and/or the flow's direction. The act of selecting records from the data store is sometimes called a "data pull".

The Partitioning Switches describe various types of traffic behavior (e.g., TCP traffic, or all traffic going to port 80). When a flow record matches all of the behaviors, it can be written to a *pass* stream (i.e., file). If a record fails to match any of these behavior predicates, it can be written to a *fail* stream. (You may also write every record **rwfilter** reads to an *all* stream.) These output streams from **rwfilter** are always binary SiLK Flow records. The output must be either written to a file or piped into another tool in the SiLK Suite, and **rwfilter** complains if it determines you are attempting to send the stream to a terminal. To view the records, pipe the records into **rwcut(1)**.

In addition to the partitioning switches built in to **rwfilter**, additional partitioning predicates can be created as C or PySiLK plug-ins, and these can be loaded into **rwfilter** using the **--plugin** and/or **--python-file** switches as described below.

Instead of using the selection switches to choose flow records from the data store, **rwfilter** can apply the partitioning switches to existing files of SiLK flow records---such as files generated by a previous invocation of **rwfilter**. To run **rwfilter** in this mode, you may

- specify, on the command line, the files and/or named pipes from which **rwfilter** should read SiLK Flow records. Specifying `stdin` or `-` or the command line causes **rwfilter** to read flow records from the standard input.
- use the **--input-pipe** switch to specify a named pipe, or specify `stdin` or `-` as the argument to this switch to have **rwfilter** read flow records from the standard input.
- use the **--xargs** switch to specify a file that contains the names of the input files to process. When **--xargs** is used without an argument, **rwfilter** attempts to read the names of the file from the standard input. The name of each input file must appear on a single line.

When **rwfilter** is reading flow records from input files, some of the selection switches act as partitioning switches. The remaining selection switches may not be specified when using the alternate forms of input, and it is an error to specify multiple types of input.

Unlike many other tools in the SiLK tool suite, **rwfilter** requires that you specify one or more Output Switches that tell **rwfilter** what types of output to produce.

Finally, there are Miscellaneous Switches that control other aspects of **rwfilter**.

Read Selection Argument Values from a File

As of SiLK 3.20, the Selection Switches **--class**, **--type**, **--flowtypes**, and **--sensors** accept a value in the form "@*PATH*", where `@` is the "at" character (ASCII 0x40) and *PATH* names a file or a path to a file. For example, the following reads the name of types from the file *t.txt* and uses the sensors `S3`, `S7`, and the names and/or IDs read from */tmp/sensor.txt*:

```
rwfilter --type=@t.txt --sensors=S3,@/tmp/sensor.txt,S7 ...
```

Multiple @*PATH* values are allowed within a single argument. If the name of the file is `-`, the names are read from the standard input.

The file must be a text file. Blank lines are ignored as are comments, which begin with the `#` character and continue to the end of the line. Whitespace at the beginning and end of a line is ignored as is whitespace that surrounds commas; all other whitespace within a line is significant.

A file may contain a value on each line and/or multiple values on a line separated by commas and optional whitespace. For example:

```
# Sensor 4
      S4
# The first sensors
S0, S1,S2
S3     # Sensor 3
```

An attempt to use an @*PATH* directive in a file is an error.

When **rwfilter** is parsing the name of a file, it converts the sequences `@,` and `@@` to `,` and `@`, respectively. For example, **--class=@cl@@ss.txt@,v** reads the class from the file *cl@ss.txt,v*. It is an error if any other character follows an embedded `@` (**--flowtypes=@f@il** contains `@i`) or if a single `@` occurs at the end of the name (**--sensor=@errat@**).

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

Selection Switches

To read files from the data store, use the following options to specify which files to process. When **rwfilter** gets its input from files listed on the command line or from the **--xargs** or **--input-pipe** switches, the first four switches (**--class**, **--type**, **--flowtypes**, and **--sensors**) act as partitioning switches, and specifying any other selection switch produces an error.

--class={*CLASS* | @*PATH*}

The **--class** switch is used to specify a group of data files to process. Only a single class may be selected with the **--class** switch; for multiple classes, use the **--flowtypes** switch. The argument may be "@*PATH*" which causes **rwfilter** to open the file *PATH* and read the class name from it; see Read Selection Argument Values from a File for details. Classes are defined in the **silk.conf(5)** site configuration file. If neither the **--class** nor **--flowtypes** option is given, the default-class as specified in *silk.conf* is used. To see the available classes and the default class, either examine the output from **rwfilter --help** or invoke **rwsiteinfo(1)** with the switch **--fields=class,default-class**.

--type={`all` | *TYPE*[,*TYPE*,@*PATH* ...]}

The **--type** predicate further specifies data within the selected *CLASS* by listing the *TYPE*s of traffic to process. The switch takes either the keyword `all` to select all types for *CLASS* or a comma-separated list of types names and "@*PATH*" directives, where @*PATH* tells **rwfilter** to read type names from the file *PATH*; see Read Selection Argument Values from a File for details. Types are defined in *silk.conf*, they typically refer to the direction of the flow, and they may vary by class. When

neither the **--type** nor **--flowtypes** switch is given, a list of default types is used: The default-type list is determined by the value of *CLASS*, and the default types often include only incoming traffic. To see the available types and the default types for each class, examine the **--help** output of **rwfilter** or run **rwsiteinfo** with **--fields=class,type,default-type**.

--flowtypes=*CLASS/TYPE*[,*CLASS/TYPE*,*@PATH* ...]

The **--flowtypes** predicate provides an alternate way to specify class/type pairs. The **--flowtypes** switch allows a single **rwfilter** invocation to process data from multiple classes. The keyword `all` may be used for the *CLASS* and/or *TYPE* to select all classes and/or types. As of SiLK 3.20.0, the arguments may also include "@*PATH*" which causes **rwfilter** to open the file *PATH* and read the class/type pairs from it; see Read Selection Argument Values from a File.

--sensors=*SENSOR*[,*SENSOR*,*SENSOR-GROUP*,*@PATH* ...]

The **--sensors** switch is used to select data from specific sensors. The parameter is a comma separated list of sensor names, sensor IDs (integers), ranges of sensor IDs, sensor group names, and/or "@*PATH*" directives. As described in Read Selection Argument Values from a File, @*PATH* tells **rwfilter** to read the names of the sensors from the file *PATH*. Sensors and sensor groups are defined in the **silk.conf(5)** site configuration file, and the **rwsiteinfo(1)** command can be used to print a mapping of sensor names to IDs and classes (**--fields=sensor,id-sensor,class:list**)). When the **--sensors** switch is not specified, the default is to use all sensors which are valid for the specified class(es).

--start-date=*YYYY/MM/DD[:HH]*

--end-date=*YYYY/MM/DD[:HH]*

The date predicates indicate which days and hours to consider when creating the list of files. The dates may be expressed as seconds since the UNIX epoch or in `YYYY/MM/DD[:HH]` format, where the hour is optional. A `T` may be used in place of the `:` to separate the day and hour. Whether the `YYYY/MM/DD[:HH]` strings represent times in UTC or the local timezone depend on how SiLK was compiled. To determine how your version of SiLK was compiled, see the `Timezone support` setting in the output from **rwfilter --version**.

When times are expressed in `YYYY/MM/DD[:HH]` format:

- When both **--start-date** and **--end-date** are specified to hour precision, all hours within that time range are processed.
- When **--start-date** is specified to day precision, the hour specified in **--end-date** (if any) is ignored, and files for all dates between midnight on **start-date** and 23:59 on **end-date** are processed.
- When **--start-date** is specified to hour precision and **--end-date** is specified to day precision, the hour of the start-date is used as the hour for the end-date.
- When **--end-date** is not specified and **--start-date** is specified to day precision, files for that complete day are processed.
- When **--end-date** is not specified and **--start-date** is specified to hour precision, files for that single hour are processed.

When at least one time is expressed as seconds since the UNIX epoch:

- When **--end-date** is specified in epoch seconds, the given **--start-date** and **--end-date** are considered to be in hour precision.
- When **--start-date** is specified in epoch seconds and **--end-date** is specified in `YYYY/MM/DD[:HH]` format, the start-date is considered to be in day precision if it divisible by 86400, and hour precision otherwise.

- When **--start-date** is specified in epoch seconds and **--end-date** is not given, the start-date is considered to be in hour-precision.

When neither **--start-date** nor **--end-date** is given, **rwfilter** processes all files for the current day.

It is an error to specify **--end-date** without specifying **--start-date**.

It is an error to specify **--start-date** when **rwfilter** believes there is some other input specified (see Non-Selection Input Switches).

--data-rootdir=*ROOT_DIRECTORY*

Tell **rwfilter** to use *ROOT_DIRECTORY* as the root of the data repository, which overrides the location given in the SILK_DATA_ROOTDIR environment variable, which in turn overrides the location that was compiled into **rwfilter** (/data). It is an error to specify this switch when files are specified on the command line or Non-Selection Input Switches are given.

--print-missing-files

This option prints to the standard error the names of the files that **rwfilter**'s file selection switches expected to find but did not. The file names are preceded by the text 'Missing '; each file name appears on a separate line. This switch is useful for debugging, but the list of files it produces can be misleading. For example, suppose there is a decommissioned sensor that still appears in the *silk.conf* file; **rwfilter** considers these data files as *missing* even though their absence is expected. Use the output from this switch judiciously. It is an error to specify this switch when files are specified on the command line or Non-Selection Input Switches are given.

Non-Selection Input Switches

Instead of using the Selection Switches to read flow records from files in the data store, you can tell **rwfilter** to process files named on the command line or use one (and only one) of the following switches. To have **rwfilter** read flow records from the standard input, specify `stdin` or `-` as the name of an input file or use the (deprecated) **--input-pipe** switch.

--xargs

--xargs=*INPUT_PATH*

Read the names of the input files from *INPUT_PATH* or from the standard input if *INPUT_PATH* is not provided. The input is expected to have one filename per line. **rwfilter** opens each named file in turn and reads records from it as if the filenames had been listed on the command line.

--input-pipe=*INPUT_PATH*

Specify a source for SiLK Flow records, where *INPUT_PATH* is a named pipe or the string `stdin` or `-` to represent the standard input. You do not need to use this switch, you can simply specify the named pipe or the strings `stdin` or `-` on the command line. **NOTE:** This switch is deprecated, and it will be removed in the SiLK 4.0 release.

Output Switches

At least one of the following output switches must be provided:

--all-destination=*ALL_PATH*

Write every SiLK Flow record to *ALL_PATH*, where *ALL_PATH* refers to a file, a named pipe, the string `stderr` to refer to the standard error, or the strings `stdout` or `-` to refer to the standard output. This switch may be repeated to write all input records to multiple locations.

--fail-destination=*FAIL_PATH*

Write SiLK Flow records that have failed ANY of the partitioning predicates to *FAIL_PATH*, where *FAIL_PATH* refers to a non-existent file, a named pipe, the string `stderr` to refer to the standard error, or the strings `stdout` or `-` to refer to the standard output. This switch may be repeated to write records that fail any predicate to multiple locations.

--pass-destination=*PASS_PATH*

Write SiLK Flow records that have passed ALL of the partitioning predicates to *PASS_PATH*, where *PASS_PATH* refers to a non-existent file, a named pipe, the string `stderr` to refer to the standard error, or the strings `stdout` or `-` to refer to the standard output. This switch may be repeated to write records that pass every predicate to multiple locations.

--print-statistics

--print-statistics=*STATS_PATH*

Print a one line summary specifying the number of files processed, the total number of records read, the number of records that passed all partitioning predicates, and the number of records that failed. If *STATS_PATH* is provided, the summary is printed there; otherwise it is printed to the standard error. This switch cannot be mixed with **--print-volume-statistics**. When running **rwfilter** with multiple threads and **--max-pass-records** or **--max-fail-records** is specified, the statistics may not match the number of records written by **rwfilter**.

--print-volume-statistics

--print-volume-statistics=*STATS_PATH*

Print a four line summary of **rwfilter**'s processing. For each of all records, records that pass all the partitioning predicates, and records that fail, print the number of flow records and the number of packets and bytes represented by those flow records. The output also includes the number of files processed. If *STATS_PATH* is provided, the summary is printed there; otherwise it is printed to the standard error. This switch cannot be mixed with **--print-statistics**. When running **rwfilter** with multiple threads and **--max-pass-records** or **--max-fail-records** is specified, the statistics may not match the number of records written by **rwfilter**.

Partitioning Switches

rwfilter supports the following partitioning switches, at least one of which must be specified (unless the only Output Switch is **--all-destination**). The switches are **AND**'ed together; i.e., to pass the filter, the record must pass the test implied by each switch. Any record that does not pass is written to the **fail-destination**(s), if specified.

Each partitioning switch defines a test. These tests can be grouped into several broad categories; within each category, the tests are applied in the order in which the switches appear on the command line. The categories of the partitioning tests are:

- tests for IP addresses (including the IPset checks), ports, protocol, times, TCP flags, byte and packet counts, IP version, application, country codes
- tests based on the **--tuple-file** switch
- tests that use the address type or prefix map mapping files
- tests that use the IP-Association plug-in

- tests based on the **--python-expr** and **--python-file** switches
- tests defined in C-plugins and loaded via **--plugin**

Partitioning Switches for IP Addresses

There are three families of switches that partition based on an IP address. Each family can partition by the source IP, the destination IP, the next hop IP, or either source or destination IP. Each family includes a **--not-*** variant to reverse the sense of the test.

The **--*cidr**-family takes as its argument an *IP_OR_CIDR_LIST*, which is a one or more of the following separated by commas: an IPv4 address (`10.1.2.3`), an IPv6 address (`2001:db8::10.1.2.3`), an unsigned 32-bit integer representing an IPv4 address (`167838211`), or any of those with a CIDR block designation (`192.168.0.0/16`, `2001:db8::/32`, `167772160/8`).

The **--*set**-family requires that you store the IPs in a binary IPset file and pass the name of the file to the switch. IPset files are created from SiLK Flow records with **rwset(1)**, or from textual input with **rwsetbuild(1)**.

The **--*address**-family (which includes **--next-hop-id**) takes as its argument a single IP address, a single CIDR block, or a single SiLK IP Wildcard. A SiLK IP Wildcard may represent multiple, disjointed IPv4 or IPv6 addresses. An IP Wildcard contains an IP in its canonical form, except each part of the IP (where *part* is an octet for IPv4 or a hexadectet for IPv6) may be a single value, a range, a comma separated list of values and ranges, or the letter `x` to signify any value for that part of the IP (that is, `0-255` for IPv4). You may not specify a CIDR suffix when using the IP Wildcard notation. The following *IP_WILDCARD*s all represent the same value:

```
::ffff:0:0/112
::ffff:0:x
::ffff:0:aaab-ffff,aaaa,0-aaa9
::ffff:0.0.0.0/112
::ffff:0.0.128-254,0-126,255,127.x
```

The next hop address often has a value of 0.0.0.0 since the default configuration of SiLK does not store the next hop address in the data repository.

With one restriction, any combination of IP partitioning switches is allowed in a single **rwfilter** invocation: A positive and negative version of the same switch (e.g., **--sipset** and **--not-sipset**) is not allowed. (**--sipset** and **--not-scidr** may be used together, as can **--sipset** and **--not-dipset**.)

The address-partitioning switches are:

--scidr=*IP_OR_CIDR_LIST*

> Pass the record if its source IP address matches a value in *IP_OR_CIDR_LIST*, a comma separated list of IPs and/or CIDR blocks. See also **--saddress** and **--sipset**.

--dcidr=*IP_OR_CIDR_LIST*

> Pass the record if its destination IP address matches a value in *IP_OR_CIDR_LIST*. See also **--daddress** and **--dipset**.

--any-cidr=*IP_OR_CIDR_LIST*

> Pass the record if either its source or its destination IP address matches a value in *IP_OR_CIDR_LIST*. This switch does *not* consider the next hop IP address. See also **--any-address** and **--anyset**.

--nhcidr=*IP_OR_CIDR_LIST*

Pass the record if its next hop IP address matches a value in *IP_OR_CIDR_LIST*. See also **--next-hop-id** and **--nhipset**.

--not-scidr=*IP_OR_CIDR_LIST*

Pass the record if its source IP address does not match a value in *IP_OR_CIDR_LIST*, a comma separated list of IPs and/or CIDR blocks. See also **--not-saddress** and **--not-sipset**.

--not-dcidr=*IP_OR_CIDR_LIST*

Pass the record if its destination IP address does not match a value in *IP_OR_CIDR_LIST*. See also **--not-daddress** and **--not-dipset**.

--not-any-cidr=*IP_OR_CIDR_LIST*

Pass the record if neither its source nor its destination IP address matches a value in *IP_OR_CIDR_LIST*. See also **--not-any-address** and **--not-anyset**.

--not-nhcidr=*IP_OR_CIDR_LIST*

Pass the record if its next hop IP address does not match a value in *IP_OR_CIDR_LIST*. See also **--not-next-hop-id** and **--not-nhipset**.

--saddress=*IP_WILDCARD*

Pass the record if its source IP address is matched by the SiLK IP Wildcard *IP_WILDCARD*. To match on multiple IPs, use **--scidr** or create an IPset and use **--sipset**.

--daddress=*IP_WILDCARD*

Pass the record if its destination IP address is matched by *IP_WILDCARD*, a SiLK IP Wildcard. See also **--dcidr** and **--dipset**.

--any-address=*IP_WILDCARD*

Pass the record if either its source or its destination IP address is matched by *IP_WILDCARD*, a SiLK IP Wildcard. This switch does *not* consider the next hop IP address. See also **--any-cidr** and **--anyset**.

--next-hop-id=*IP_WILDCARD*

Pass the record if its next hop IP address is matched by this *IP_WILDCARD*, a SiLK IP Wildcard. To match on multiple IPs, use **--nhcidr** or create an IPset and use **--nhipset**.

--not-saddress=*IP_WILDCARD*

Pass the record if its source IP address is *not* matched by this *IP_WILDCARD*, a SiLK IP Wildcard. See also **--not-scidr** and **--not-sipset**.

--not-daddress=*IP_WILDCARD*

Pass the record if its destination IP address is *not* matched by this *IP_WILDCARD*. See also **--not-dcidr** and **--not-dipset**.

--not-any-address=*IP_WILDCARD*

Pass the record if neither its source nor its destination IP address is matched by this *IP_WILDCARD*. Does *not* consider the next hop address. See also **--not-any-cidr** and **--not-anyset**.

--not-next-hop-id=*IP_WILDCARD*

Pass the record if its next hop IP address is *not* matched by this *IP_WILDCARD*. See also **--not-nhcidr** and **--not-nhipset**.

--sipset=*IP_SET_FILENAME*

Pass the record if its source IP address is in the list of IPs contained in the binary set file *IP_SET_FILENAME*. See also **--scidr**.

--dipset=*IP_SET_FILENAME*

As **--sipset** for the destination IP address. See also **--dcidr**.

--anyset=*IP_SET_FILENAME*

Pass the record if either its source IP address or its destination IP address is in the list of IPs contained in the binary set file *IP_SET_FILENAME*. Does *not* consider the next hop IP. See also **--any-cidr**.

--nhipset=*IP_SET_FILENAME*

As **--sipset** for the next-hop IP address. See also **--nhcidr**.

--not-sipset=*IP_SET_FILENAME*

Pass the record if its source IP address is *not* in the list of IPs contained in the binary set file *IP_SET_FILENAME*. See also **--not-scidr**.

--not-dipset=*IP_SET_FILENAME*

As **--not-sipset** for the destination IP address. See also **--not-dcidr**.

--not-anyset=*IP_SET_FILENAME*

Pass the record if neither its source IP address nor its destination IP address is in the list of IPs contained in the binary set file *IP_SET_FILENAME*. Does *not* consider the next hop IP. See also **--not-any-cidr**.

--not-nhipset=*IP_SET_FILENAME*

As **--not-sipset** for the next hop IP address. See also **--not-nhcidr**.

Partitioning Switches for Remainder of Five-Tuple

The following switches partition based on the protocol and source or destination port. The parameter to each of these switches is an *INTEGER_LIST*, which is a comma-separated list of individual non-negative integer values and ranges of those values. For example, `1,2,3,5-10,99-103`. A range may be specified without an upper limit, such as `1-`, in which case the upper limit is set to the maximum value.

--sport=*INTEGER_LIST*

Pass the record if its source port is in this *INTEGER_LIST*, possible values are 0-65535.

--dport=*INTEGER_LIST*

Pass the record if its destination port is in this *INTEGER_LIST*, possible values are 0-65535

--aport=*INTEGER_LIST*

Pass the record if its source port and/or its destination port is in this *INTEGER_LIST*, possible values are 0-65535. For example, use **--aport**=*25* to see all SMTP conversions regardless or where they originated.

--protocol=*INTEGER_LIST*

Pass the record if its IP Suite Protocol is in this *INTEGER_LIST*, possible values are 0-255.

--icmp-type=*INTEGER_LIST*

Pass the record if its ICMP (or ICMPv6) type is in this *INTEGER_LIST*; possible values 0-255. This switch also verifies that the flow's protocol is 1 (or 58 if the flow is IPv6). It is an error to specify a **--protocol** that does not include 1 and/or 58.

--icmp-code=*INTEGER_LIST*

Pass the record if its ICMP (or ICMPv6) code is in this *INTEGER_LIST*; possible values 0-255. This switch also verifies that the flow's protocol is 1 (or 58 if the flow is IPv6). It is an error to specify a **--protocol** that does not include 1 and/or 58.

Partitioning Switches for Time

These switches partition based on whether the time stamps on the flow record occur within the specified time window. The form of the argument is range of two dates, start-window and end-window, each in the form `YYYY/MM/DD[:HH[:MM[:SS[.ssssss]]]]`, for example `2003/01/31:23:45:00.000-2003/01/31:23:59:59.999` represents the last fifteen minutes of Jan 31, 2003. (A `T` may be used in place of `:` to separate the day and hour.) The start-window and end-window must be set to at least day precision. For the start-window, unspecified hour, minute, second, and millisecond values are set to 0; for the end-window, those values are set to 23, 59, 59, and 999 respectively. Thus `2003/01/31:23-2003/01/31:23` becomes `2003/01/31:23:00:00.000-2003/01/31:23:59:59.999`. If an end-window is not given, it is set to the start-window, giving a window of a single millisecond. The date strings are considered to be in the timezone specified when SiLK was compiled, which you can determine from the output of **rwfilter --version**. You may also specify the times as seconds since the UNIX epoch; when the end-time is in epoch seconds, an unspecified milliseconds value is set to 999 and otherwise the value is unchanged.

--active-time=*TIME_WINDOW*

Pass the record if the record was active at ANY time during this *TIME_WINDOW*. If a single time is specified, pass the record if it was active at that instant.

--stime=*TIME_WINDOW*

Pass the record if its starting time is in this *TIME_WINDOW*.

--etime=*TIME_WINDOW*

As **--stime** for the ending time.

--duration=*DECIMAL_RANGE*

Pass the record if its duration--that is, the record's end time minus its start time, as measured in seconds--is in this *DECIMAL_RANGE*. Use floating point numbers to specify millisecond values. The range should be specified as *MIN-MAX*; for example, `5.0-10.031`. If a single value is given, the duration must match that value exactly. The upper limit may be omitted; for example, a range of `1.5-` passes records whose duration is at least 1.5 seconds.

Partitioning Switches for Volume

The following switches partition based on the volume of the flow; that is, the number of bytes or packets. For additional volume-related switches, load the **flowrate** plug-in as described in the **flowrate(3)** manual page.

These switches accept a range of non-negative integers or decimal values. If the upper limit is omitted, the volume must be at least that size. If the argument is a single value, the volume must match that value exactly.

--bytes=*INTEGER_RANGE*

Pass the record if its byte count is in this *INTEGER_RANGE*.

--packets=*INTEGER_RANGE*

Pass the record if its packet count is in this *INTEGER_RANGE*.

--bytes-per-packet=*DECIMAL_RANGE*

Pass the record if its average bytes per packet count (bytes/packet) is in this *DECIMAL_RANGE*.

Partitioning Switches for TCP Flags

When a flow generator creates a flow record from TCP packets, it creates a field that is the bit-wise OR of the TCP flags from all packets that comprise that flow record. Some flow generators, such as **yaf(1)**, can export two TCP flag fields: one contains the flags on the first packet in the flow, and the second contains the bit-wise OR of the remaining packets.

To partition records based on their TCP flags values, there is a recommended set of switches and legacy-supported switches. The switches accept the following letters to represent the named TCP flag: `F`=FIN; `S`=SYN; `R`=RST; `P`=PSH; `A`=ACK; `U`=URG; `E`=ECE; `C`=CWR. As of SiLK 3.20.0, the symbol `-` is accepted and represents all TCP flags (`FSRPAUEC`).

The recommended set of switches take a comma separated list of pairs of TCP flags, where the pair is separated by a slash (`/`). The value to the left of the slash is the *HIGH_SET* and it must be a subset of the value to the right of the slash, which is the *MASK_SET*. For a record to pass the filter, the flags in the *HIGH_SET* must be on and the remaining flags in *MASK_SET* must be off. Flags not in *MASK_SET* may have any value. If a list of pairs is given, the record passes if any pair in the list matches. For example, `--flags-all=S/S,A/A` passes flows that have either the SYN or the ACK flag set, `--flags-all=S/SA` passes flow records where SYN is high and ACK is low, and `--flags-all=/F` passes flows where FIN is off. This list of flag pairs is called a *HIGH_MASK_FLAGS_LIST*.

The recommended switches for TCP flag partitioning are:

--flags-all=*HIGH_MASK_FLAGS_LIST*

Pass the record if any of the *HIGH_SET*/*MASK_SET* pairs is true when looking at the bit-wise OR of the TCP flags across **all** packets in the flow.

--flags-initial=*HIGH_MASK_FLAGS_LIST*

As **--flags-all**, except this switch considers only the initial packet in the flow, for flow generators that can generate that field.

--flags-session=*HIGH_MASK_FLAGS_LIST*

As **--flags-all**, except this switch considers the bit-wise OR of the TCP flags across the second through the final packet in the flow; that is, ignoring the flags on the first packet.

The TCP-flag partitioning switches supported for legacy reasons are:

--tcp-flags=*TCP_FLAGS*

Pass the record if, for any one of its packets, *any* of the specified *TCP_FLAGS* was set (high), where *TCP_FLAGS* contains the letters `F`,`S`,`R`,`P`,`A`,`U`,`E`,`C`. For example, **--tcp-flags=ASF** passes records where ACK is set, or SYN is set, or FIN is set (this is equivalent to **--flags-all=A/A,S/S,F/F**).

--ack-flag={0|1}

Pass the record when either the argument is 1 and the ACK flag is set (high) or the argument is 0 and the ACK flag is unset (low).

--cwr-flag={0|1}

Pass the record when either the argument is 1 and the CWR flag is set (high) or the argument is 0 and the CWR flag is unset (low).

--ece-flag={0|1}

Pass the record when either the argument is 1 and the ECE flag is set (high) or the argument is 0 and the ECE flag is unset (low).

--fin-flag={0|1}

Pass the record when either the argument is 1 and the FIN flag is set (high) or the argument is 0 and the FIN flag is unset (low).

--psh-flag={0|1}

Pass the record when either the argument is 1 and the PSH flag is set (high) or the argument is 0 and the PSH flag is unset (low).

--rst-flag={0|1}

Pass the record when either the argument is 1 and the RST flag is set (high) or the argument is 0 and the RST flag is unset (low).

--syn-flag={0|1}

Pass the record when either the argument is 1 and the SYN flag is set (high) or the argument is 0 and the SYN flag is unset (low).

--urg-flag={0|1}

Pass the record when either the argument is 1 and the URG flag is set (high) or the argument is 0 and the URG flag is unset (low).

Partitioning Switches for Other Flow Characteristics

Other than the **--ip-version** switch, the fields queried by the following switches may always be zero. The default configuration of SiLK does not store the fields that contain the SNMP values. The other fields are not present in NetFlow v5, and require use of properly-configured enhanced collection software, such as **yaf(1)**, http://tools.netsa.cert.org/yaf/.

--ip-version={4|6|4,6}

Passes the record if its IP Version is in the specified list. This switch determines how IPv4 and IPv6 flow records are handled when SiLK has been compiled with IPv6 support. When the argument to this switch is `4`, **rwfilter** writes records marked as IPv6 to the fail-destination, regardless of the IP addresses it contains. When the argument to this switch is `6`, **rwfilter** writes records marked as IPv4 to the fail-destination. When SiLK has not been compiled with IPv6 support, the only legal value for this switch is `4`, and any IPv6 flows in the input ignored (that is, they are not written to either the pass-destination nor the fail-destination).

--application=*INTEGER_LIST*

Some flow generation software can inspect the contents of the packets that comprise a flow and use traffic signatures to label the content of the flow. SiLK calls this label the *application*; **yaf** refers to

it as the *appLabel* (see the **applabel(1)** manual page in the yaf distribution). The application value is the port number that is traditionally used for that type of traffic (see the */etc/services* file on most UNIX systems). For example, traffic that the flow generator recognizes as FTP has a value of 21, even if that traffic is being routed through the standard HTTP/web port (80). The flow generator uses a value for 0 if the application cannot be determined. The **--application** switch passes the flow if the flow's application value is in the specified *INTEGER_LIST*, which is a comma separated list of integers from 0 to 65535 inclusive and ranges of those integers. The list of valid appLabels is determined by your site's **yaf** installation.

--attributes=*ATTRIBUTES_LIST*

The *attributes* field in SiLK Flow records describes characteristics about how the flow record was generated or about the packets that comprise the flow record. The *ATTRIBUTES_LIST* argument is similar to the *HIGH_MASK_FLAGS_LIST* argument to the **--flags-all** switch. *ATTRIBUTES_LIST* is a comma separated list of up to 8 *HIGH_ATTRIBUTES*/*MASK_ATTRIBUTES* pairs, where *HIGH_ATTRIBUTES* and *MASK_ATTRIBUTES* are strings of the characters `S`,`T`,`C`,`F`, and *HIGH_ATTRIBUTES* is a subset of *MASK_ATTRIBUTES*. As of SiLK 3.20.0, the symbol `-` is accepted and represents all attributes (`STCF`). **rwfilter** passes the record if, for any pair of attributes in the list, the attributes listed in *HIGH_ATTRIBUTES* are set and the remaining attributes in *MASK_ATTRIBUTES* are not-set. The valid *attributes* are:

`S`

All the packets in this flow record are exactly the same size.

`T`

The flow generator prematurely created a record for a long-lived session due to the connection's lifetime reaching the *active timeout* of the flow generator. (Also, when **yaf** is run with the **--silk** switch, it prematurely creates a flow and marks it with `T` if the byte count of the flow cannot be stored in a 32-bit value.)

`C`

The flow generator created this flow as a continuation of long-running connection, where the previous flow for this connection met a timeout.

`F`

The flow generator saw additional packets in this flow following a packet with the FIN flag set (excluding ACK packets).

For a long-lived connection spanning several flow records, the first flow record is marked with a `T` indicating that it hit the active timeout. The second through next-to-last records are marked with `CT` indicating that the flow is a continuation of a connection that timed out and that this flow also timed out. The final flow is marked with a `C`, indicating that it was created as a continuation of an active flow.

--input-index=*INTEGER_LIST*

Pass the record if its `in` field is in this *INTEGER_LIST*, which is a comma separated list of integers from 0 to 65535, inclusive, and ranges of those integers. When present, the `in` field normally contains the incoming SNMP interface, but it may contain the vlanId if the packing tools were configured to capture it (see **sensor.conf(5)**).

--output-index=*INTEGER_LIST*

Pass the record if its `out` field is in this *INTEGER_LIST*. When present, the `out` field normally contains the outgoing SNMP interface, but it may contain the postVlanId if the packing tools were configured to capture it.

--any-index=*INTEGER_LIST*

Pass the record if its `in` field or if its `out` field is in this *INTEGER_LIST*.

Selection Switches Acting as Partitioning Switches

The following four switches are normally file selection switches that select which files **rwfilter** reads within the data repository. However, when **rwfilter** gets input without querying the data repository (that is, from files listed on the command line, from files specified by **--xargs**, or from the **--input-pipe**), these switches become partitioning switches and determine whether a record is written to the pass-destination or fail-destination.

--class={*CLASS* | @*PATH*}

Pass the record if its class is *CLASS* or the class named in *PATH* and its type is listed in the **--type** switch, or its type is in the default type list for *CLASS* when **--type** is not specified. Examine the output of **rwfilter --help** to see the list of available classes, available types, and their default values, or use **rwsiteinfo --fields=class,type,mark-defaults**.

--flowtypes=*CLASS*/*TYPE*[,*CLASS*/*TYPE*,@*PATH* ...]

Pass the record its if class/type value is one of those listed or read from the file *PATH*. The keyword `all` may be used for the *CLASS* and/or *TYPE* to select all classes and/or types. This switch cannot be used when either **--class** or **--type** is used. Use **rwfilter --help** or **rwsiteinfo --fields=class,type** to see the list of available classes and types.

--sensors=*SENSOR*[,*SENSOR*,*SENSOR-GROUP*,@*PATH* ...]

Pass the record if its sensor is one of those listed. The parameter is a comma separated list of sensor names, sensor IDs (integers), ranges of sensor IDs, sensor group names, and "@*PATH*" directives. Use the **rwsiteinfo(1)** command to see the list of sensors. Support for sensor group names was added in SiLK 3.21.0.

--type={`all` | *TYPE*[,*TYPE*,@*PATH* ...]}

Pass the record if its type is one of those listed or read from the file *PATH* and its class is specified by **--class**, or its class is the default class when the **--class** switch is not specified. Use **rwfilter --help** to see the list of available classes, available types, and their defaults, or use **rwsiteinfo --fields=class,type,mark-defaults**.

Partitioning Switches that use Additional Mapping Files

Additional partitioning switches are available that allow one to partition flow records depending on a label, where the label is computed from an IP address or port on the record and an additional mapping file.

--pmap-file=*PATH*

--pmap-file=*MAPNAME*:*PATH*

Load the prefix map file located at *PATH* and create partitioning switches named **--pmap-src-*map-name***, **--pmap-dst-*map-name***, and **--pmap-any-*map-name*** where *map-name* is either the *MAPNAME* part of the argument or the map-name specified when the file was created (see **rwpmapbuild(1)**). If no map-name is available, **rwfilter** creates switch names as described below (**--pmap-saddress**, **--pmap-sport-proto**, etc). Specify *PATH* as `-` or `stdin` to read from the standard input. The switch may be repeated to load multiple prefix map files; each file must have a unique map-name. The **--pmap-file** switch(es) must precede all other **--pmap-*** switches. For more information, see **pmapfilter(3)**.

--pmap-src-*map-name*=*LABELS*

If the prefix map associated with *map-name* is an IP prefix map, this matches records with a source IPv4 address that maps to a label contained in the list of labels in *LABELS*. If the prefix map associated with *map-name* is a proto-port prefix map, this matches records with a protocol and source port combination that maps to a label contained in the list of labels in *LABELS*.

--pmap-dst-*map-name*=*LABELS*

Similar to **--pmap-src-*map-name***, but uses the destination IP or the protocol and destination port.

--pmap-any-*map-name*=*LABELS*

If the prefix map associated with *map-name* is an IP prefix map, this matches records with a source IP address or a destination IP address that maps to a label contained in the list of labels in *LABELS*. If the prefix map associated with *map-name* is a port/protocol prefix map, this matches records with a protocol and source port or destination port combination that maps to a label contained in the list of labels in *LABELS*.

--pmap-saddress=*LABELS*

--pmap-daddress=*LABELS*

--pmap-any-address=*LABELS*

These are deprecated switches created by **pmapfilter** that correspond to **--pmap-src-*map-name***, **--pmap-dst-*map-name***, and **--pmap-any-*map-name***, respectively. These switches are available when an IP prefix map is used that is not associated with a map-name.

--pmap-sport-proto=*LABELS*

--pmap-dport-proto=*LABELS*

--pmap-any-port-proto=*LABELS*

These are deprecated switches created by **pmapfilter** that correspond to **--pmap-src-*map-name***, **--pmap-dst-*map-name***, and **--pmap-any-*map-name***, respectively. These switches are available when a proto-port prefix map is used that is not associated with a map-name.

--scc=*COUNTRY_CODE_LIST*

--dcc=*COUNTRY_CODE_LIST*

--any-cc=*COUNTRY_CODE_LIST*

Pass the record if one its IP addresses maps to a country code that is specified in *COUNTRY_CODE_LIST*. For **--scc**, the source IP must match. For **--dcc**, the destination IP must match. For **--any-cc**, either the source or the destination must match. *COUNTRY_CODE_LIST* is a comma separated list of lowercase two-letter country codes---defined by ISO 3166-1 (see for example https://www.iso.org/iso-3166-country-codes.html or https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2)---as well as the following special codes:

`--`

N/A (e.g. private and experimental reserved addresses)

`a1`

anonymous proxy

`a2`

satellite provider

`o1`
: other

For example: `cx,uk,kr,jp,--`. To use this switch, the country code mapping file must be available in the default location, or in the location specified by the SILK_COUNTRY_CODES environment variable. See **ccfilter(3)** for details.

--stype={0|1|2|3}

--dtype={0|1|2|3}

Pass a flow record depending on whether the IP address is internal, external, or non-routable. These switches use the mapping file specified by the SILK_ADDRESS_TYPES environment variable, or the *address_types.pmap* mapping file, as described in **addrtype(3)**. When the parameter is 0, pass the record if its source (**--stype**) IP address or destination (**--dtype**) IP address is non-routable. When 1, pass if internal. When 2, pass if external (i.e., routable but not internal). When 3, pass if not internal (non-routable or external).

Partitioning Switches across Multiple Fields

The **--tuple-*** family of switches allows the user to partition flow records based on multiple values of the five-tuple.

--tuple-file=*TUPLE_FILENAME*

This switch provides support for partitioning by arbitrary subsets of the basic five-tuple:

```
{source-ip,destination-ip,source-port,destination-ip-port,protocol}
```

A SiLK Flow record passes the test when the record's fields match one of the tuples; if the SiLK record does not match any tuple, the record fails. The tuples are read from the text file *TUPLE_FILENAME* which must contain lines of delimited fields. The default delimiter is |, but may be specified with the **--tuple-delimiter** switch. Each field contains one member of the tuple; the fields may appear in any order. The fields may represent any subset of the five-tuple, but each line in the file must define the same subset. A field that is present but has no value generates an error. If you want the field to match any value, it is best that you not include that field in your input.

In addition to the tuple-lines, *TUPLE_FILENAME* may contain blank lines and comments (which begin with `#` and continue to the end of the line). The first line of *TUPLE_FILENAME* may contain a title labeling the fields in the file. This title line is ignored when the **--tuple-fields** switch is given.

The IP fields may contain an IPv4 address, an integer, or a IP in CIDR block notation. Comma-separated lists (`80,443`) and ranges (`0-1023,8080`) are supported for the ports and protocol fields. **NOTE:** Currently the code is not clever in its support for CIDR notation and ranges in that each occurrence is fully expanded. When this occurs, the memory required to hold the search tree quickly grows.

--tuple-fields=*FIELDS*

FIELDS contains the list of fields (columns) to parse from the *TUPLE_FILENAME* in the order in which they appear in the file. When this switch is not provided, **rwfilter** treats the first line in *TUPLE_FILENAME* as a title line and attempts to determine the fields (a la **rwtuc(1)**); **rwfilter** exits if it cannot determine the fields.

FIELDS is a comma separated list of field-names, field-integers, and ranges of field-integers; a range is specified by separating the start and end of the range with a hyphen (`-`). Names can be abbreviated to their shortest unique prefix. The field names and their descriptions are:

sIP,sip,1
: source IP address

dIP,dip,2
: destination IP address

sPort,sport,3
: source port

dPort,dport,4
: destination port

protocol,5
: IP protocol

--tuple-direction=*DIRECTION*

Allows you to change the comparison between the tuple and the SiLK Flow record. This switch allows one to look for traffic in the reverse direction (or both directions) without having to write all of the rules twice. The available directions are:

forward
: The tuple's fields are compared against the corresponding fields on the flow; that is, sIP is compared with sIP, dIP with dIP, sPort with sPort, dPort with dPort, and protocol with protocol. This is the default.

reverse
: The tuple's fields are compared against the opposite fields on the flow; that is, sIP is compared with dIP, dIP with sIP, sPort with dPort, dPort with sPort, and protocol with protocol.

both
: Both of the above comparisons are performed.

--tuple-delimiter=*CHAR*

Specifies the character separating the input fields. When the switch is not provided, the default of | is used.

Partitioning Switches that use the PySiLK Plug-in

The SiLK Python plug-in provides support for filtering by expressions or complex functions written in the Python programming language. See the **silkpython(3)** and **pysilk(3)** manual pages for information and examples for how to use Python to manipulate SiLK data structures. When multiple Partitioning Switches are given, the Python plug-in is the next-to-last to be invoked. Only the code specified by the **--plugin** switch is called after the Python code.

--python-file=*FILENAME*

Pass the record if the result of the processing the flow with the function named **rwfilter()** in *FILENAME* is true. The function should take a single **silk.RWRec** object as an argument. See **silkpython(3)** for details.

--python-expr=*PYTHON_EXPRESSION*

Pass the record if the result of the processing the flow with the specified *PYTHON_EXPRESSION* is true. The expression is evaluated as if it appeared in the following context:

```
from silk import *
def rwfilter(rec):
    return (PYTHON_EXPRESSION)
```

Partitioning Switches that use the IP-Association Plug-In

The IPA plug-in, *ipafilter.so*, provides switches that can partition flows using data in an IP Association database. For this plug-in to be available, SiLK must be compiled with IPA support and IPA must be configured. See **ipafilter(3)** and http://tools.netsa.cert.org/ipa/ for additional information.

--ipa-src-expr=*IPA_EXPR*

Use *IPA_EXPR* to partition flows based on the source IP of the flow matching the *IPA_EXPR* expression.

--ipa-dst-expr=*IPA_EXPR*

Use *IPA_EXPR* to partition flows based on the destination IP of the flow matching the *IPA_EXPR* expression.

--ipa-any-expr=*IPA_EXPR*

Use *IPA_EXPR* to partition flows based on either the source or destination IP of the flow matching the *IPA_EXPR* expression.

Miscellaneous Switches

--compression-method=*COMP_METHOD*

Specify the compression library to use when writing output files. If this switch is not given, the value in the SILK_COMPRESSION_METHOD environment variable is used if the value names an available compression method. When no compression method is specified, output to the standard output or to named pipes is not compressed, and output to files is compressed using the default chosen when SiLK was compiled. The valid values for *COMP_METHOD* are determined by which external libraries were found when SiLK was compiled. To see the available compression methods and the default method, use the **--help** or **--version** switch. SiLK can support the following *COMP_METHOD* values when the required libraries are available.

none

Do not compress the output using an external library.

zlib

Use the **zlib(3)** library for compressing the output, and always compress the output regardless of the destination. Using zlib produces the smallest output files at the cost of speed.

lzo1x

Use the *lzo1x* algorithm from the LZO real time compression library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead.

snappy

Use the *snappy* library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead. *Since SiLK 3.13.0.*

best

Use lzo1x if available, otherwise use snappy if available, otherwise use zlib if available. Only compress the output when writing to a file.

--dry-run

Perform a sanity check on the input arguments to check that the arguments are acceptable. In addition, prints to the standard output the names of the files that would be accessed (and the names of missing

files if **--print-missing** is specified). **rwfglob(1)** can also be used to generate the lists of files that **rwfilter** would access.

--help

Print the available options and exit. Options that add fields (for example, options that load plug-ins, prefix maps, or PySiLK extensions) can be specified before the **--help** switch so that the new options appear in the output. The available classes and types are included in output; you may specify a different root directory or site configuration file before **--help** to see the classes and types available for that site.

--max-fail-records=*N*

Write *N* records to each **--fail-destination**. **rwfilter** stops reading input once it has written these *N* records unless **--pass-destination** or **--all-destination** switch(es) are also specified.

--max-pass-records=*N*

Write *N* records to each **--pass-destination**. **rwfilter** stops reading input once it has written these *N* records unless **--fail-destination** or **--all-destination** switch(es) are also specified.

--note-add=*TEXT*

Add the specified *TEXT* to the header of the output file as an annotation. This switch may be repeated to add multiple annotations to a file. To view the annotations, use the **rwfileinfo(1)** tool.

--note-file-add=*FILENAME*

Open *FILENAME* and add the contents of that file to the header of the output file as an annotation. This switch may be repeated to add multiple annotations. Currently the application makes no effort to ensure that *FILENAME* contains text; be careful that you do not attempt to add a SiLK data file as an annotation.

--plugin=*PLUGIN*

Augment the partitioning switches by using run-time loading of the plug-in (shared object) whose path is *PLUGIN*. The switch may be repeated to load multiple plug-ins. The creation of plug-ins is described in the **silk-plugin(3)** manual page. When multiple partitioning switches are given, the code specified by the **--plugin** switch(es) is last to be invoked. When *PLUGIN* does not contain a slash (**/**), **rwfilter** attempts to find a file named *PLUGIN* in the directories listed in the FILES section. If **rwfilter** finds the file, it uses that path. If *PLUGIN* contains a slash or if **rwfilter** does not find the file, **rwfilter** relies on your operating system's **dlopen(3)** call to find the file. When the SILK_PLUGIN_DEBUG environment variable is non-empty, **rwfilter** prints status messages to the standard error as it attempts to find and open each of its plug-ins.

--print-filenames

Print the names of input files as they are read. This can be useful feedback for a long-running **rwfilter** process.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwfilter** searches for the site configuration file in the locations specified in the FILES section.

--threads=*N*

Invoke **rwfilter** with *N* threads reading the input files. When this switch is not provided, the value in the SILK_RWFILTER_THREADS environment variable is used. If that variable is not set, **rwfilter** runs with a single thread. Using multiple threads, performance of **rwfilter** is greatly improved for queries that look at many files but return few records. Preliminary testing has found that performance peaks around four threads per CPU, but performance varies depending on the type of query and the number of records returned.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

The most basic filtering involves looking at specific traffic over a specific time. For example:

```
$ rwfilter --start-date=2003/02/19:00 --end-date=2003/02/19:23     \
       --proto=6 --pass-destination=tcp-in.rw
```

creates a file, *tcp-in.rw* containing all **incoming** TCP traffic on February 19, 2003. The **--start-date** and **--end-date** switches select which files to examine. The **--proto** switch partitions the flow records into a *pass* stream (records whose protocol is 6---that is, TCP) and a *fail* stream (all other records). The **--pass-destination** switch (often shortened to **--pass**) tells **rwfilter** to write the records that pass the **--proto** test to the file *tcp-in.rw*.

The *tcp-in.rw* file contains SiLK Flow data in a binary format. To examine the contents, use the command **rwcut(1)**. This query only selects incoming traffic because the **silk.conf(5)** configuration file at most sites tells **rwfilter** to look at incoming traffic unless an explicit **--type** switch is given.

The following query gets all TCP traffic (for the default class) for February 19, 2003.

```
$ rwfilter --type=all --start-date=2003/02/19  \
       --proto=6 --pass-destination=alltcp.rw
```

Note the addition of **--type=all**. This query also relies on the default behavior of **--start-date** to consider a full day's worth of data when no hour is specified.

The above query gets all traffic for the default class. If your *silk.conf* file has a single class, that query captures all of it. For *silk.conf* files that specify multiple classes, the following gets all TCP traffic for February 19, 2003:

```
$ rwfilter --flowtypes=all/all --start-date=2003/02/19     \
       --proto=6 --pass-destination=alltcp.rw
```

To get all non-TCP traffic, there are two approaches. **rwfilter** does not supply a way to choose a negated set of protocols, but you can choose all protocols other than TCP:

```
$ rwfilter --start-date=2003/02/19:00 --end-date=2003/02/19:23     \
       --proto=0-5,7-255 --pass-destination=non-tcp.rw
```

The other approach is to use the **--fail-destination** switch (often shortened to **--fail**) that contains the records that failed one or more of the partitioning test(s):

```
$ rwfilter --start-date=2003/02/19:00 --end-date=2003/02/19:23     \
       --proto=6 --fail-destination=non-tcp.rw
```

To print information about the number of flow records that pass a filter, use **--print-volume-statistics**. This can be combined with other output switches.

```
$ rwfilter --start-date=2003/02/19:00 --end-date=2003/02/19:23     \
       --proto=6 --print-volume-stat --pass-destination=tcp-in.rw
     |        Recs|     Packets|         Bytes|  Files|
Total|      515359|     2722887|    1343819719|    180|
 Pass|      512071|     2706571|    1342851708|       |
 Fail|        3288|       16316|        968011|       |
```

If you want to see the number of records in a file produced by **rwfilter**, or to remind yourself how a file was created, use **rwfileinfo(1)**:

```
$ rwfileinfo tcp-in.rw
tcp-in.rw:
  format(id)          FT_RWGENERIC(0x16)
  version             16
  byte-order          littleEndian
  compression(id)     lzo1x(2)
  header-length       208
  record-length       52
  record-version      5
  silk-version        2.4.0
  count-records       512071
  file-size           8576160
  command-lines
      1  rwfilter --start-date=2003/02/19:00 --end-date=2003/02/19:23 \
           --proto=6 --print-volume-stat --pass-destination=tcp-in.rw
```

Once a file is written, **rwfilter** can process the file again. Traffic on port 25 is most likely email (SMTP) traffic. To split the email traffic from the other traffic, use:

```
$ rwfilter --aport=25 --pass=mail.rw --fail=not-mail.rw tcp-in.rw
```

This command puts traffic where the source or destination port was 25 into the file *mail.rw*, and all other traffic into the file *not-mail.rw*. The **--fail-destination** is an effective way to reverse the sense of a test. For example, to remove traffic on port 80 from the *not-mail.rw* file, run the command:

```
$ rwfilter --aport=80 --fail=not-mail-web.rw not-mail.rw
```

To verify that the *not-mail-web.rw* file does not contain any traffic on ports 25 or 80, you can use the **--print-statistics** switch and see that 0 records pass:

```
$ rwfilter --aport=25,80 --print-stat not-mail-web.rw
Files     1.  Read    54641.  Pass        0. Fail     54641.
```

The file maintains a history of the commands that created it:

```
$ rwfileinfo not-mail-web.rw
not-mail-web.rw:
  format(id)          FT_RWGENERIC(0x16)
  version             16
  byte-order          littleEndian
  compression(id)     lzo1x(2)
  header-length       364
  record-length       52
  record-version      5
  silk-version        2.4.0
  count-records       54641
  file-size           762875
  command-lines
      1  rwfilter --start-date=2003/02/19:00 --end-date=2003/02/19:23 \
           --proto=6 --print-volume-stat --pass-destination=tcp-in.rw
      2  rwfilter --aport=25 --pass=mail.rw --fail=not-mail.rw        \
           tcp-in.rw
      3  rwfilter --aport=80 --fail=not-mail-web.rw not-mail.rw
```

The following finds all outgoing traffic from February 19, 2003, going to an external email server. Traffic going to a server contacts that server on its well-known port, and the flow record's destination port should hold that well-known port:

```
$ rwfilter --type=out --start-date=2003/02/19 --print-volume-stat  \
       --dport=25 --proto=6
```

To limit the result to completed connections, select flow records that contain at least three packets, use the **--packets** switch with an open-ended range:

```
$ rwfilter --type=out --start-date=2003/02/19 --print-volume-stat  \
       --dport=25 --proto=6 --packets=3-
```

To limit the search to a particular internal CIDR block, 10.1.2.0/24, there are three different IP-partitioning switches you can use. The final approach uses **rwsetbuild(1)** to create an IPset file from textual input.

```
$ rwfilter --type=out --start-date=2003/02/19 --print-volume-stat  \
       --dport=25 --proto=6 --packets=3- --scidr=10.1.2.0/24

$ rwfilter --type=out --start-date=2003/02/19 --print-volume-stat  \
       --dport=25 --proto=6 --packets=3- --saddress=10.1.2.x

$ echo "10.1.2.0/24" | rwsetbuild > my-set.set
$ rwfilter --type=out --start-date=2003/02/19 --print-volume-stat  \
       --dport=25 --proto=6 --packets=3- --sipset=my-set.set
```

rwfilter does not have to output its records to a file; instead, the output from **rwfilter** can be piped into a another SiLK tool. You must still use the **--pass-destination** switch (or **--fail-destination** or **--all-destination** switch), but by providing the argument of `stdout` or `-` to the switch you tell **rwfilter** to write its output to the standard output.

For example, to get the IPs of the external email servers that the monitored network contacted, pipe the **rwfilter** output into **rwset(1)**, and tell **rwset** to store the destination addresses:

```
$ rwfilter --type=out --start-date=2003/02/19 --dport=25          \
       --proto=6 --packets=3- --scidr=10.1.2.0/24 --pass=stdout   \
  | rwset --dip-file=external-mail-servers.set
```

rwfilter can also pipe its output as input to another **rwfilter** command, which allows them to be chained together. **rwfilter** does not read from the standard input by default; you must explicitly give `stdin` or `-` as the stream to read:

```
$ rwfilter --type=out,outweb --start-date=2003/02/19              \
       --scidr=10.1.2.0/24 --pass=stdout                          \
  | rwfilter --proto=17 --pass=udp.rw --fail=stdout stdin         \
  | rwfilter --proto=6 --pass=stdout --fail=non-tcp-udp.rw stdin  \
  | rwfilter --aport=25 --pass=mail.rw --fail=stdout stdin        \
  | rwfilter --aport=80,443 --pass=web.rw                         \
       --fail=tcp-non-web-mail.rw stdin
```

This chain of commands looks at outgoing traffic on February 19, 2003, originating from the internal net-block 10.1.2.0/24, creates the following files:

udp.rw

Outgoing UDP traffic

non-tcp-udp.rw

Outgoing traffic that is neither TCP nor UDP

mail.rw

Outgoing TCP traffic on port 25, most of which is probably email (SMTP). Since the query looks at outgoing traffic and the **--aport** switch was used, this file represents email going from the internal 10.1.2.0/24 to external mail servers, and the responses from any internal mail servers that exist in the 10.1.2.0/24 net-block to external clients.

web.rw

Outgoing TCP traffic on ports 80 and 443, most of which is probably web traffic (HTTP,HTTPS). As with the *mail.rw* file, this file represents queries to external web servers and responses from internal web servers.

tcp-non-web-mail.rw

Outgoing TCP traffic other than that on ports 25, 80, and 443

Expert users can create even more complicated chains of **rwfilter** commands using named pipes.

ENVIRONMENT

SILK_RWFILTER_THREADS

The number of threads to use while reading input files or files selected from the data store.

PYTHONPATH

This environment variable is used by Python to locate modules. When **--python-file** or **--python-expr** is specified, **rwfilter** must load the Python files that comprise the PySiLK module, such as *silk/__init__.py*. If this *silk/* directory is located outside Python's normal search path (for example, in the SiLK installation tree), it may be necessary to set or modify the PYTHONPATH environment variable to include the parent directory of *silk/* so that Python can find the PySiLK module.

SILK_PYTHON_TRACEBACK

When set, Python plug-ins output traceback information on Python errors to the standard error.

SILK_COUNTRY_CODES

This environment variable allows the user to specify the country code mapping file that the **--scc** and **--dcc** switches use. The value may be a complete path or a file relative to the SILK_PATH. See the FILES section for standard locations of this file.

SILK_ADDRESS_TYPES

This environment variable allows the user to specify the address type mapping file that the **--stype** and **--dtype** switches use. The value may be a complete path or a file relative to the SILK_PATH. See the FILES section for standard locations of this file.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_COMPRESSION_METHOD

This environment variable is used as the value for **--compression-method** when that switch is not provided. *Since SiLK 3.13.0.*

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. This value overrides the compiled-in value, and **rwfilter** uses it unless the **--data-rootdir** switch is specified. In addition, **rwfilter** may use this value when searching for the SiLK site configuration files. See the FILES section for details.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files and plug-ins, **rwfilter** may use this environment variable. See the FILES section for details.

TZ

When a SiLK installation is built to use the local timezone (to determine if this is the case, check the `Timezone support` value in the output from **rwfilter --version**), the value of the TZ environment variable determines the timezone in which **rwfilter** parses timestamps. If the TZ environment variable is not set, the default timezone is used. Setting TZ to 0 or the empty string causes timestamps to be parsed as UTC. The value of the TZ environment variable is ignored when the SiLK installation uses utc. For system information on the TZ variable, see **tzset(3)** or **environ(7)**.

SILK_PLUGIN_DEBUG

When set to 1, **rwfilter** prints status messages to the standard error as it attempts to find and open each of its plug-ins.

SILK_LOGSTATS

When set to a non-empty value, **rwfilter** treats the value as the path to an external program to execute with information about this **rwfilter** invocation. If the value in SILK_LOGSTATS does not contain a slash or if it references a file that does not exist, is not a regular file, or is not executable, the SILK_LOGSTATS value is silently ignored. The arguments to the external program are:

- The application name, i.e., `rwfilter`. Note that `rwfilter` is always used as this argument, regardless of the name of the executable.
- The version number of this command line, currently `v0001`.
- The start time of this invocation, as seconds since the UNIX epoch.
- The end time of this invocation, as seconds since the UNIX epoch.
- The number of data files opened for reading.
- The number of records read.
- The number of records written.
- A variable number of arguments that are the complete command line used to invoke **rwfilter**, including the name of the executable.

SILK_LOGSTATS_RWFILTER

If set, this environment variable overrides the value specified in SILK_LOGSTATS.

SILK_LOGSTATS_DEBUG

If the environment variable is set to a non-empty value, **rwfilter** prints messages to the standard error about the SILK_LOGSTATS value being used and either the reason why the value cannot be used or the arguments to the external program being executed.

FILES

${SILK_ADDRESS_TYPES}

${SILK_PATH}/share/silk/address_types.pmap

${SILK_PATH}/share/address_types.pmap

/usr/local/share/silk/address_types.pmap

/usr/local/share/address_types.pmap

Possible locations for the address types mapping file required by the **--stype** and **--dtype** switches.

${SILK_CONFIG_FILE}

ROOT_DIRECTORY/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided, where *ROOT_DIRECTORY/* is the directory **rwfilter** is using as the root of the data repository.

${SILK_COUNTRY_CODES}

${SILK_PATH}/share/silk/country_codes.pmap

${SILK_PATH}/share/country_codes.pmap

/usr/local/share/silk/country_codes.pmap

/usr/local/share/country_codes.pmap

Possible locations for the country code mapping file required by the **--scc** and **--dcc** switches.

${SILK_DATA_ROOTDIR}/

/data/

Locations for the root directory of the data repository when the **--data-rootdir** switch is not specified.

${SILK_PATH}/lib64/silk/

${SILK_PATH}/lib64/

${SILK_PATH}/lib/silk/

${SILK_PATH}/lib/

/usr/local/lib64/silk/

/usr/local/lib64/

/usr/local/lib/silk/

/usr/local/lib/

Directories that **rwfilter** checks when attempting to load a plug-in.

NOTES

The ability to use @*PATH* in **--class**, **--type**, **--flowtypes**, and **--sensors** was added in SiLK 3.20.0.

As of SiLK 3.20.0, **--types** is an alias for **--type**.

The **--sensors** switch also accepts the names of groups defined in the **silk.conf(5)** file as of SiLK 3.21.0.

rwfilter is the most commonly used application in the suite. It provides access to the data files and performs all the basic queries.

rwfilter supports a variety of I/O options - in addition to reading from the data store, **rwfilter** results can be chained together with named pipes to output results to multiple files simultaneously. An introduction to named pipes is outside the scope of this document, however.

Two often underused options are **--dry-run** and **--print-statistics**. **--dry-run** performs a sanity check on the arguments and can be used, especially for complicated arguments, to check that the arguments are acceptable. **--print-statistics** used without **--pass-destination** or **--fail-destination** simply prints aggregate statistics to the standard error on a single line, and it can be used to do a quick pass through the data to get aggregate counts before going in deeper into the phenomenon being investigated.

--print-filename can be used as a progress meter; during long jobs, it shows which file is currently being read by **rwfilter**. **--print-filename** does not provide meaningful feedback with piped input.

Filters are applied in the order given on the command line. It is best to apply the biggest filters first.

The **rwfilter** command line is written into the header of the output file(s). You may use the **rwfileinfo(1)** command to see this information.

SEE ALSO

rwcut(1), **rwfglob(1)**, **rwfileinfo(1)**, **rwset(1)**, **rwtuc(1)**, **rwsetbuild(1)**, **rwsiteinfo(1)**, **rwpmapbuild(1)**, **addrtype(3)**, **ccfilter(3)**, **flowrate(3)**, **ipafilter(3)**, **pmapfilter(3)**, **pysilk(3)**, **silkpython(3)**, **silk-plugin(3)**, **silk.conf(5)**, **sensor.conf(5)**, **silk(7)**, **rwflowpack(8)**, **yaf(1)**, **applabel(1)**, **zlib(3)**, **dlopen(3)**, **tzset(3)**, **environ(7)**, *Analysts' Handbook: Using SiLK for Network Traffic Analysis*

rwgeoip2ccmap

Create a country code prefix map from a GeoIP Legacy file

SYNOPSIS

```
rwgeoip2ccmap [--input-path=PATH] [--output-path=PATH] [--dry-run]
      [--mode={[auto] [ipv4|ipv6] [csv|binary] [geoip2|legacy]}]
      [--fields=FIELDS] [--note-add=TEXT] [--note-file-add=FILENAME]
      [--invocation-strip]

rwgeoip2ccmap --help

rwgeoip2ccmap --version
```

Legacy Synopsis

```
rwgeoip2ccmap {--csv-input | --v6-csv-input | --encoded-input}
      [--input-file=PATH] [--output-file=PATH] [--dry-run]
      [--note-add=TEXT] [--note-file-add=FILENAME]
      [--invocation-strip]
```

DESCRIPTION

Prefix maps provide a way to map field values to string labels based on a user-defined map file. The country code prefix map, typically named *country_codes.pmap*, is a special prefix map that maps an IP address to a two-letter country code as defined by ISO 3166 part 1. For additional information, see https://www.iso.org/iso-3166-country-codes.html and https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2.

rwgeoip2ccmap creates the country code prefix map by reading one of the country code database files distributed by MaxMind(R) http://www.maxmind.com/. **rwgeoip2ccmap** supports these formats:

- GeoIP2 or GeoLite2 Country comma-separated value (CSV) files. Set **--input-path** to the name of the directory containing the files *GeoIP2-Country-Blocks-IPv4.csv*, *GeoIP2-Country-Blocks-IPv6.csv*, and *GeoIP2-Country-Locations-en.csv* (or the *GeoLite2-*.csv* versions of those files). *Since SiLK 3.17.0.*
- GeoIP2 or GeoLite2 Country binary file. Set **--input-path** to the *GeoIP2-Country.mmdb* of *GeoLite2-Country.mmdb* file. **Note:** This requires that SiLK was compiled with support for the `libmaxminddb` library. *Since SiLK 3.17.0.*
- GeoIP or GeoLite Legacy Country Code binary file, either IPv4 or IPv6. Set **--input-path** to *GeoIP.dat* or *GeoIPv6.dat*. You may also pipe or redirect the file into **rwgeoip2ccmap**'s standard input.
- GeoIP or GeoLite Legacy Country Code comma-separated value (CSV) file, either IPv4 or IPv6. Set **--input-path** to *GeoIPCountryWhois.csv* or *GeoIPv6.csv*. You may also pipe or redirect the file into **rwgeoip2ccmap**'s standard input.

The GeoIP2 and GeoLite2 files provide up to three GeoName codes for each network block, where the GeoName may represent a country (and its continent) or only a continent.

location

The country where the network is located.

registered

The country in which the ISP has registered the network.

represented

The country that is represented by users of the network (consider an overseas military base).

As of SiLK 3.17.2, the **--fields** switch allows you to select the order in which these values are checked.

See the EXAMPLES section below for the details on how to convert these files to a SiLK country-code prefix map file.

The country code prefix map file is used to map IP addresses to country codes in various SiLK tools as documented in the **ccfilter(3)** man page. As a brief overview, you may

- partition by an IP address's country code in **rwfilter(1)**
- display an IP address's country code in **rwcut(1)**
- sort by an IP address's country code in **rwsort(1)**
- bin by an IP address's country code in **rwstats(1)**, **rwuniq(1)**, and **rwgroup(1)**.

Use **rwpmapcat(1)** with the **--country-codes** switch to print the contents of a country code prefix map.

The **rwpmaplookup(1)** command can use the country code mapping file to display the country code for textual IP addresses.

To create a general prefix map file from textual input, use **rwpmapbuild(1)**.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--mode==*MODE_OPTIONS*

Specify the type of the input and whether **rwgeoip2ccmap** creates a prefix map containing IPv4 or IPv6 addresses. *MODE_OPTIONS* is a comma-separated list of the following values. When not specified, *MODE_OPTIONS* defaults to **auto**. *Since SiLK 3.12.0; changed in SiLK 3.17.0.*

auto

Determine the type of the input based on the argument to **--input-path**, and determine the type of prefix map to create based on the IP addresses that appear on the first line of input for a CSV file or by the depth of the tree for a binary input file. This is the default mode.

ipv6

Create an IPv6 prefix map. When reading CSV input, the IPv4 addresses are mapped into the ::ffff:0:0/96 netblock. This value may not be combined with **ipv4**.

ipv4

Create an IPv4 prefix map. When reading CSV input, the IPv6 addresses in the ::ffff:0:0/96 netblock are mapped to IPv4 addresses and all other IPv6 addresses are ignored. When reading GeoIP2 binary data, the IPv6 addresses in the ::0:0/96 netblock are mapped to IPv4. This value may not be combined with **ipv6**.

csv

Read textual input containing IP addresses in a comma separated value format. and create an IPv4 prefix map. Any IPv6 addresses in the ::ffff:0:0/96 netblock are mapped to an IPv4 address and all other IPv6 addresses are ignored. This value may not be combined with **binary**. *Since SiLK 3.17.0.*

binary

Read a MaxMind binary country code database file in the GeoIP Legacy, GeoIP2, or GeoLite2 formats. Support for the GeoIP2 formats requires that SiLK was built with libmaxminddb support. This value may not be combined with **csv**.

geoip2

Expect the input to be the GeoIP2 or GeoLite2 country code formats (CSV or binary). GeoIP2/GeoLite2 data may not be read from the standard input. The value may not be combined with **legacy**. *Since SiLK 3.17.0.*

legacy

Expect the input to be the GeoIP Legacy country code format (CSV or binary). This mode is enabled if the input is being read from the standard input. This value may not be combined with **geoip2**. *Since SiLK 3.17.0.*

--input-path=*PATH*

Read the comma-separated value (CSV) or binary forms of the GeoIP2, GeoLite2, GeoIP Legacy, or GeoLite Legacy country code database from *PATH*. For GeoIP2 data, the **--input-path** switch is required, and it must either be the location of the *GeoLite2-Country.mmdb* file for binary data or the **directory** containing the *GeoLite2-Country-Blocks-IPv4.csv* file for CSV data. **rwgeoip2ccmap** supports reading GeoIP Legacy data (either binary or CSV) from the standard input. You may use `stdin` or `-` to represent the standard input; when this switch is not provided, the input is read from the standard input unless the standard input is a terminal. **rwgeoip2ccmap** reads read textual input from the terminal if the standard input is explicitly specified as the input. (Added in SiLK 3.17.0 as a replacement for **--input-file**.)

--output-path=*PATH*

Write the binary country code prefix map to *PATH*. You may use `stdout` or `-` to represent the standard output. When this switch is not provided, the prefix map is written to the standard output unless the standard output is connected to a terminal. (Added in SiLK 3.17.0 as a replacement for **--output-file**.)

--dry-run

Check the syntax of the input file and do not write the output file. *Since SiLK 3.12.0.*

--fields=*FIELDS*

Select which of the GeoName fields are used when processing a GeoIP2 or GeoLite2 file, given that these files provide up to three GeoName values for each IP block, some GeoName values map to a continent but not a specific country, and some blocks are flagged as being by an anonymizing proxy or a satellite provider. (For details on the content of the files, see https://dev.maxmind.com/geoip/geoip2/geoip2-city-country-csv-databases/.)

FIELDS is a comma-separated list of one or more of the following values. **rwgeoip2ccmap** checks each value and stops when it finds one that is non-empty. If all are empty, no mapping is added for

the IP block. When the switch not given, the default is "location, registered, represented, continent, flags". *Since SiLK 3.17.2.*

The supported field values and their mapping to the fields in the GeoIP2 files are:

location
The country where the IP address block is located. (geoname_id)

registered
The country in which the ISP has registered the IP address block. (registered_country_geoname_id)

represented
The country that is represented by users of the IP address block (consider an overseas military base). (represented_country_geoname_id)

flags
Whether the IP is marked as being used by an anonymizing proxy or a satellite provider. (is_anonymous_proxy, is_satellite_provider)

continent
For binary GeoIP2 files, the continent code. For CSV GeoIP2 files, if this appears before `location`, `registered`, and `represented`, **rwgeoip2ccmap** uses the first of those fields that is non-empty and maps to either a country or a continent. If this appears after those fields, **rwgeoip2ccmap** uses the first non-empty field that maps to a country and only when none map to a country does **rwgeoip2ccmap** check those fields for a continent code.

--note-add=*TEXT*

Add the specified *TEXT* to the header of the output file as an annotation. This switch may be repeated to add multiple annotations to a file. To view the annotations, use the **rwfileinfo(1)** tool. *Since SiLK 3.12.0.*

--note-file-add=*FILENAME*

Open *FILENAME* and add the contents of that file to the header of the output file as an annotation. This switch may be repeated to add multiple annotations. Currently the application makes no effort to ensure that *FILENAME* contains text; be careful that you do not attempt to add a SiLK data file as an annotation. *Since SiLK 3.12.0.*

--invocation-strip

Do not record the command used to create the prefix map in the output. When this switch is not given, the invocation is written to the file's header, and the invocation may be viewed with **rwfileinfo(1)**. *Since SiLK 3.12.0.*

--help

Print the available options and exit.

--version

Print the version number and exit the application.

Deprecated Options

The following switches are deprecated.

--csv-input

Assume the input is the CSV GeoIP Legacy country code data for IPv4. Use **--mode=ipv4,csv,legacy** as the replacement. Deprecated as of SiLK 3.12.0.

--v6-csv-input

Assume the input is the CSV GeoIP Legacy country code data for IPv6. Use **--mode=ipv6,csv,legacy** as the replacement. Deprecated as of SiLK 3.12.0.

--encoded-input

Assume the input is the specially-encoded binary form of the GeoIP Legacy country code data for either IPv4 or IPv6. Use **--mode=binary,legacy** as the replacement. Deprecated as of SiLK 3.12.0.

--input-file=*PATH*

Read the input from *PATH*. An alias for **--input-path**. Added in SiLK 3.12.0; deprecated as of SiLK 3.17.0.

--output-file=*PATH*

Write the binary country code prefix map to *PATH*. An alias for **--output-path**. Added in SiLK 3.12.0; deprecated as of SiLK 3.17.0.

EXAMPLES

The following examples show how to create the country code prefix map file, *country_codes.pmap*, from various forms of input. Once you have created the *country_codes.pmap* file, you should copy it to */usr/local/share/silk/country_codes.pmap* so that the **ccfilter(3)** plug-in can find it. Alternatively, you can set the SILK_COUNTRY_CODES environment variable to the location of the *country_codes.pmap* file.

In these examples, the dollar sign (**$**) represents the shell prompt. Some input lines are split over multiple lines in order to improve readability, and a backslash (\) is used to indicate such lines.

MaxMind GeoIP2 or GeoLite2 Comma Separated Values Files

Download the CSV version of the MaxMind GeoIP2 or GeoLite2 country database file, e.g., *GeoLite2-Country-CSV_20180327.zip*. This archive is created with the **zip(1)** utility and contains a directory of multiple files. Expand the archive with **unzip(1)**:

```
$ unzip GeoLite2-Country-CSV_20180327.zip
Archive:  GeoLite2-Country-CSV_20180327.zip
  inflating: GeoLite2-Country-CSV_20180327/GeoLite2-Country-...
  ...
```

rwgeoip2ccmap uses three of those files:

GeoLite2-Country-Blocks-IPv4.csv

A mapping from IPv4 netblocks to a `geoname_id`s.

GeoLite2-Country-Blocks-IPv6.csv

A mapping from IPv6 netblocks to a `geoname_id`s.

GeoLite2-Country-Locations-en.csv

A mapping from `geoname_id`s to continent and country.

Run **rwgeoip2ccmap** and set **--input-path** to the name of the directory.

```
$ rwgeoip2ccmap --input-path=GeoLite2-Country-CSV_20180327 \
        --output-path=country_codes.pmap
```

MaxMind GeoIP2 or GeoLite2 Binary File

Support for reading GeoIP2 binary files requires that **rwgeoip2ccmap** was compiled with support for the `libmaxminddb` library.

Download the binary version of the MaxMind GeoIP2 or GeoLite2 country database file, e.g., *GeoLite2-Country_20180327.tar.gz*. The file is a compressed (**gzip(1)**) tape archive (**tar(1)**). Most versions of the **tar** program allow you to expand the archive using

```
$ tar zxf GeoLite2-Country_20180327.tar.gz
```

Older versions of **tar** may require you to invoke **gzip** yourself

```
$ gzip -d -c GeoLite2-Country_20180327.tar.gz \
  | tar cf -
```

The result is a directory named *GeoLite2-Country_20180327* or similar.

Run **rwgeoip2ccmap** and set **--input-path** to the name of the *GeoLite2-Country.mmdb* file in the directory.

```
$ rwgeoip2ccmap \
       --input-path=GeoLite2-Country_20180327/GeoLite2-Country.mmdb \
       --output-path=country_codes.pmap
```

MaxMind Legacy IPv4 Comma Separated Values File

Download the CSV version of the MaxMind GeoIP Legacy Country database for IPv4, *GeoIPCountryCSV.zip*. Running **unzip -l** on the zip file should show a single file, *GeoIPCountryWhois.csv*.) To expand this file, use the **unzip(1)** utility.

```
$ unzip GeoIPCountryCSV.zip
Archive:  GeoIPCountryCSV.zip
  inflating: GeoIPCountryWhois.csv
```

Create the *country_codes.pmap* file by running

```
$ rwgeoip2ccmap --input-path=GeoIPCountryWhois.csv \
       --output-path=country_codes.pmap
```

You may avoid creating the GeoIPCountryWhois.csv file by using the **-p** option of **unzip** to pass the output of **unzip** directly to **rwgeoip2ccmap**:

```
$ unzip -p GeoIPCountryCSV.zip  \
  | rwgeoip2ccmap --mode=ipv4 --output-path=country_codes.pmap
```

MaxMind Legacy IPv6 Comma Separated Values File

If you download the IPv6 version of the MaxMind GeoIP Legacy Country database, use the following command to create the *country_codes.pmap* file:

```
$ gzip -d -c GeoIPv6.csv.gz  \
  | rwgeoip2ccmap --mode=ipv6 > country_codes.pmap
```

Since the *GeoIPv6.csv.gz* file only contains IPv6 addresses, the resulting *country_codes.pmap* file will display the unknown value (`--`) for any IPv4 address. See the next example for a solution.

MaxMind Legacy IPv6 and IPv4 Comma Separated Values Files

To create a *country_codes.pmap* mapping file that supports both IPv4 and IPv6 addresses, download both of the Legacy CSV files (*GeoIPv6.csv.gz* and *GeoIPCountryCSV.zip*) from MaxMind.

You need to uncompress both files and feed the result as a single stream to the standard input of **rwgeoip2ccmap**. This can be done in a few commands:

```
$ gzip -d GeoIPv6.csv.gz
$ unzip GeoIPCountryCSV.zip
$ cat GeoIPv6.csv GeoIPCountryWhois.csv  \
  | rwgeoip2ccmap --mode=ipv6 > country_codes.pmap
```

Alternatively, if your shell supports it, you may be able to use a subshell to avoid having to store the uncompressed data:

```
$ ( gzip -d -c GeoIPv6.csv.gz ; unzip -p GeoIPCountryCSV.zip )  \
  | rwgeoip2ccmap --mode=ipv6 > country_codes.pmap
```

Printing the Contents of the Country Code Prefix Map

To print the contents of a file that **rwgeoip2ccmap** creates, use the **rwpmapcat(1)** command, and specify the **--country-codes** switch:

```
$ rwpmapcat --country-codes=country_codes.pmap | head -5
           ipBlock|value|
         0.0.0.0/7|   --|
        2.0.0.0/14|   --|
        2.4.0.0/15|   --|
        2.6.0.0/17|   --|
```

To reduce the number of lines in the output by combining CIDR blocks into IP ranges, use the **--no-cidr-blocks** switch:

```
$ rwpmapcat --country-codes=country_codes.pmap --no-cidr-blocks \
  | head -5
        startIP|          endIP|value|
        0.0.0.0|     2.6.190.55|   --|
```

```
     2.6.190.56|     2.6.190.63|   gb|
     2.6.190.64|  2.255.255.255|   --|
        3.0.0.0|    4.17.135.31|   us|
```

To skip IP blocks that are unassigned and have the label `--`, use the **--ignore-label** switch:

```
$ rwpmapcat --country-codes=country_codes.pmap --ignore-label=-- \
  | head -5
           ipBlock|value|
     2.6.190.56/29|   gb|
         3.0.0.0/8|   us|
        4.0.0.0/12|   us|
       4.16.0.0/16|   us|
```

To print the contents of the default country code prefix map, specify **--country-codes** without an argument:

```
$ export SILK_COUNTRY_CODES=country_codes.pmap
$ rwpmapcat --country-codes --ignore-label=-- | head -5
           ipBlock|value|
     2.6.190.56/29|   gb|
         3.0.0.0/8|   us|
        4.0.0.0/12|   us|
       4.16.0.0/16|   us|
```

If you print the output of **rwgeoip2ccmap** without using the **--country-codes** switch, the numerical values are not decoded to characters and the output resembles the following:

```
$ rwpmapcat --no-cidr-blocks country_codes.pmap | head -5
        startIP|          endIP|     value|
        0.0.0.0|     2.6.190.55|     11565|
     2.6.190.56|     2.6.190.63|     26466|
     2.6.190.64|  2.255.255.255|     11565|
        3.0.0.0|    4.17.135.31|     30067|
```

Getting the Country Code for a Specific IP Address

Use **rwpmaplookup(1)** to get the country code for specific IP address(es). Use the **--no-files** switch when specific the IP addresses on the command line; otherwise **rwpmaplookup** treats its arguments as text files containing IP addresses. The **--country-code** switch is required for the prefix map's data to be interpreted correctly. Give an argument to the switch for a specific file, or no argument to use the default country code prefix map.

```
$ rwpmaplookup --country-codes=country_codes.pmap --no-files \
       3.4.5.6 4.5.6.7
           key|value|
        3.4.5.6|   us|
        4.5.6.7|   us|
```

```
$ export SILK_COUNTRY_CODES=country_codes.pmap
$ cat ips.txt
3.4.5.6
4.5.6.7
$ rwpmaplookup --country-codes ips.txt
            key|value|
        3.4.5.6|   us|
        4.5.6.7|   us|
```

Converting a Country Code Prefix Map to a Normal Map

The SiLK tools support using only a single country code mapping file. There may be occasions where you want to use multiple country code mapping files; for example, to see changes in an IP block's country over time, or to build separate files for each of GeoIP2 fields (location, registered, represented). One way to do this is loop through the files setting the SILK_COUNTRY_CODES environment variable to each filename and running the SiLK commands. An alternative approach is to convert the country code mapping files to ordinary prefix map files and leverage the SiLK tools' support for using multiple prefix map files in a single command.

To convert a country-code prefix map to an ordinary prefix map, use **rwpmapcat(1)** to print the contents of the country code prefix map file as text, and then use **rwpmapbuild(1)** to convert the text to an ordinary prefix map.

First, create a text file where you define a name for this prefix map, specify the mode (as either `ipv4` or `ipv6`), and specify the `default` value to be `--`:

```
$ cat /tmp/mymap.txt
map-name  cc-old
mode      ipv4
default   --
$
```

Append the output of **rwpmapcat** to this file, using the space character as the delimiter.

```
$ rwpmapcat --no-title --delimited=' ' --ignore-label=-- \
       --country-codes=country_codes.pmap                \
  >> /tmp/mymap.txt
$ head -5 /tmp/mymap.txt
map-name  cc-old
mode      ipv4
default   --
2.6.190.56/29 gb
3.0.0.0/8 us
```

Use **rwpmapbuild** to create the prefix map and save it as *cc-old.pmap*:

```
$ rwpmapbuild --input-path=/tmp/mymap.txt --output-path=cc-old.pmap
```

Use **rwfileinfo(1)** to check the map-name for the prefix map file.

```
$ rwfileinfo --fields=prefix-map cc-old.pmap
cc-old.pmap:
  prefix-map          v1: cc-old
```

Use **rwpmapcat** to view its contents:

```
$ rwpmapcat --ignore-label=-- --no-cidr-blocks cc-old.pmap | head -5
        startIP|          endIP|label|
     2.6.190.56|     2.6.190.63|   gb|
        3.0.0.0|    4.17.135.31|   us|
    4.17.135.32|    4.17.135.63|   ca|
    4.17.135.64|   4.17.142.255|   us|
```

You can use the **--pmap-file** switch of various SiLK tools to load and use the *cc-old.pmap* prefix map file (see **pmapfilter(3)** for usage).

For example, suppose you have the file *data.rw* of SiLK Flow data:

```
$ rwcut --fields=sip,dip --ipv6-policy=ignore data.rw
            sIP|            dIP|
        3.4.5.6|        4.5.6.7|
        4.5.6.7|        3.4.5.6|
```

To map the source IP addresses in the file *data.rw* using the prefix map file (the `src-cc-old` field) and a country code file (the `scc` field) with **rwcut(1)**:

```
$ export SILK_COUNTRY_CODES=country_codes.pmap
$ rwcut --pmap-file cc-old.pmap --ipv6-policy=ignore \
       --fields=sip,src-cc-old,scc data.rw
            sIP|src-cc-old|scc|
        3.4.5.6|        us| us|
        4.5.6.7|        us| us|
```

ENVIRONMENT

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SEE ALSO

ccfilter(3), **rwpmaplookup(1)**, **rwfilter(1)**, **rwcut(1)**, **rwsort(1)**, **rwstats(1)**, **rwuniq(1)**, **rwgroup(1)**, **rwpmapbuild(1)**, **rwfileinfo(1)**, **pmapfilter(3)**, **silk(7)**, **gzip(1)**, **zip(1)**, **unzip(1)**, https://dev.maxmind.com/geoip/geoip2/geolite2/, https://dev.maxmind.com/geoip/legacy/geolite/, https://www.iso.org/iso-3166-country-codes.html, https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2

NOTES

Support for GeoIP2 and GeoLite2 input files were added in SiLK 3.17.0.

Support for the binary form of the GeoIP Legacy format was removed in SiLK 3.12.0 and restored in SiLK 3.12.2.

MaxMind, GeoIP, and related trademarks are the trademarks of MaxMind, Inc.

rwgroup

Tag similar SiLK records with a common next hop IP value

SYNOPSIS

```
rwgroup
      {--id-fields=KEY | --delta-field=FIELD --delta-value=DELTA}
      [--objective] [--summarize] [--rec-threshold=THRESHOLD]
      [--group-offset=IP]
      [--note-add=TEXT] [--note-file-add=FILE] [--output-path=PATH]
      [--copy-input=PATH] [--compression-method=COMP_METHOD]
      [--site-config-file=FILENAME]
      [--plugin=PLUGIN [--plugin=PLUGIN ...]]
      [--python-file=PATH [--python-file=PATH ...]]
      [--pmap-file=MAPNAME:PATH [--pmap-file=MAPNAME:PATH ...]]
      [FILE]

rwgroup [--pmap-file=MAPNAME:PATH [--pmap-file=MAPNAME:PATH ...]]
      [--plugin=PLUGIN ...] [--python-file=PATH ...] --help

rwgroup [--pmap-file=MAPNAME:PATH [--pmap-file=MAPNAME:PATH ...]]
      [--plugin=PLUGIN ...] [--python-file=PATH ...] --help-fields

rwgroup --version
```

DESCRIPTION

rwgroup reads *sorted* SiLK Flow records (c.f. **rwsort(1)**) from the standard input or from a *single* file name listed on the command line, marks records that form a *group* with an identifier in the Next Hop IP field, and prints the binary SiLK Flow records to the standard output. In some ways **rwgroup** is similar to **rwuniq(1)**, but **rwgroup** writes SiLK flow records instead of textual output.

Two SiLK records are defined as being in the same group when the fields specified in the **--id-fields** switch match exactly and when the field listed in the **--delta-field** matches within the value given by the **--delta-value** switch. Either **--id-fields** or **--delta-fields** is required; both may be specified. A **--delta-value** must be given when **--delta-fields** is present.

The first group of records gets the identifier 0, and **rwgroup** writes that value into each record's Next Hop IP field. The ID for each subsequent group is incremented by 1. The **--group-offset** switch may be used to set the identifier of the initial group.

The **--rec-threshold** switch may be used to only write groups that contain a certain number of records. The **--summarize** switch attempts to merge records in the same group to a single output record.

rwgroup requires that the records are sorted on the fields listed in the **--id-fields** and **--delta-fields** switches. For example, a call using

```
rwgroup --id-field=2 --delta-field=9 --delta-value=3
```

should read the output of

```
rwsort --field=2,9
```

otherwise the results are unpredictable.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

At least one value for **--id-field** or **--delta-field** must be provided; **rwgroup** terminates with an error if no fields are specified.

--id-fields=*KEY*

KEY contains the list of flow attributes (a.k.a. fields or columns) that must match exactly for flows to be considered part of the same group. Each field may be specified once only. *KEY* is a comma separated list of field-names, field-integers, and ranges of field-integers; a range is specified by separating the start and end of the range with a hyphen (**-**). Field-names are case insensitive. Example:

```
--id-fields=stime,10,1-5
```

There is no default value for the **--id-fields** switch.

The complete list of built-in fields that the SiLK tool suite supports follows, though note that not all fields are present in all SiLK file formats; when a field is not present, its value is 0.

sIP,1
: source IP address

dIP,2
: destination IP address

sPort,3
: source port for TCP and UDP, or equivalent

dPort,4
: destination port for TCP and UDP, or equivalent

protocol,5
: IP protocol

packets,pkts,6
: packet count

bytes,7
: byte count

flags,8
: bit-wise OR of TCP flags over all packets

sTime,9
: starting time of flow (seconds resolution)

duration,10
: duration of flow (seconds resolution)

eTime,11
: end time of flow (seconds resolution)

sensor,12
: name or ID of sensor at the collection point

class,20
: class of sensor at the collection point

type,21
: type of sensor at the collection point

iType
: the ICMP type value for ICMP or ICMPv6 flows and zero for non-ICMP flows. Internally, SiLK stores the ICMP type and code in the `dPort` field, so there is no need have both `dPort` and `iType` or `iCode` in the sort key. This field was introduced in SiLK 3.8.1.

iCode
: the ICMP code value for ICMP or ICMPv6 flows and zero for non-ICMP flows. See note at `iType`.

icmpTypeCode,25
: equivalent to `iType`,`iCode` in **--id-fields**. This field may not be mixed with `iType` or `iCode`, and this field is deprecated as of SiLK 3.8.1. As of SiLK 3.8.1, `icmpTypeCode` may no longer be used as the argument to **--delta-field**; the `dPort` field will provide an equivalent result as long as the input is limited to ICMP flow records.

Many SiLK file formats do not store the following fields and their values will always be 0; they are listed here for completeness:

in,13
: router SNMP input interface or vlanId if packing tools were configured to capture it (see **sensor.conf(5)**)

out,14
: router SNMP output interface or postVlanId

SiLK can store flows generated by enhanced collection software that provides more information than NetFlow v5. These flows may support some or all of these additional fields; for flows without this additional information, the field's value is always 0.

initialFlags,26
: TCP flags on first packet in the flow

sessionFlags,27
: bit-wise OR of TCP flags over all packets except the first in the flow

attributes,28
: flow attributes set by the flow generator:

  `S`
  : all the packets in this flow record are exactly the same size

  `F`
  : flow generator saw additional packets in this flow following a packet with a FIN flag (excluding ACK packets)

  `T`
  : flow generator prematurely created a record for a long-running connection due to a timeout. (When the flow generator **yaf(1)** is run with the **--silk** switch, it will prematurely create a flow and mark it with `T` if the byte count of the flow cannot be stored in a 32-bit value.)

`C`

flow generator created this flow as a continuation of long-running connection, where the previous flow for this connection met a timeout (or a byte threshold in the case of **yaf**).

Consider a long-running ssh session that exceeds the flow generator's *active* timeout. (This is the active timeout since the flow generator creates a flow for a connection that still has activity). The flow generator will create multiple flow records for this ssh session, each spanning some portion of the total session. The first flow record will be marked with a `T` indicating that it hit the timeout. The second through next-to-last records will be marked with `TC` indicating that this flow both timed out and is a continuation of a flow that timed out. The final flow will be marked with a `C`, indicating that it was created as a continuation of an active flow.

application,29

guess as to the content of the flow. Some software that generates flow records from packet data, such as **yaf**, will inspect the contents of the packets that make up a flow and use traffic signatures to label the content of the flow. SiLK calls this label the *application*; **yaf** refers to it as the *appLabel*. The application is the port number that is traditionally used for that type of traffic (see the */etc/services* file on most UNIX systems). For example, traffic that the flow generator recognizes as FTP will have a value of 21, even if that traffic is being routed through the standard HTTP/web port (80).

The following fields provide a way to label the IPs or ports on a record. These fields require external files to provide the mapping from the IP or port to the label:

sType,16

categorize the source IP address as `non-routable`, `internal`, or `external` and group based on the category. Uses the mapping file specified by the SILK_ADDRESS_TYPES environment variable, or the *address_types.pmap* mapping file, as described in **addrtype(3)**.

dType,17

as **sType** for the destination IP address

scc,18

the country code of the source IP address. Uses the mapping file specified by the SILK_COUNTRY_CODES environment variable, or the *country_codes.pmap* mapping file, as described in **ccfilter(3)**.

dcc,19

as **scc** for the destination IP

src-*map-name*

label contained in the prefix map file associated with *map-name*. If the prefix map is for IP addresses, the label is that associated with the source IP address. If the prefix map is for protocol/port pairs, the label is that associated with the protocol and source port. See also the description of the **--pmap-file** switch below and the **pmapfilter(3)** manual page.

dst-*map-name*

as **src-*map-name*** for the destination IP address or the protocol and destination port.

sval

as **src-*map-name*** when no map-name is associated with the prefix map file

dval

as **dst-*map-name*** when no map-name is associated with the prefix map file

Finally, the list of built-in fields may be augmented by the run-time loading of PySiLK code or plug-ins written in C (also called shared object files or dynamic libraries), as described by the **--python-file** and **--plugin** switches.

--delta-field=*FIELD*

Specify a single field that can differ by a specified delta-value among the SiLK records that make up a group. The *FIELD* identifiers include most of those specified for **--id-fields**. The exceptions are that plug-in fields are not supported, nor are fields that do not have numeric values (e.g., class, type, flags). The most common value for this switch is `stime`, which allows records that are identical in the **id-fields** but temporally far apart to be in different groups. The switch takes a single argument; multiple delta fields cannot be specified. When this switch is specified, the **--delta-value** switch is required.

--delta-value=*DELTA_VALUE*

Specify the acceptable difference between the values of the **--delta-field**. The **--delta-value** switch is required when the **--delta-field** switch is provided. For fields other than those holding IPs, when two consecutive records have values less than or equal to *DELTA_VALUE*, the records are considered members of the same group. When the delta-field refers to an IP field, *DELTA_VALUE* is the number of **least** significant bits of the IPs to **remove** before comparing them. For example, when **--delta-field=sIP --delta-value=8** is specified, two records are the same group if their source IPv4 addresses belong to the same /24 or if their source IPv6 addresses belong to the same /120. The **--objective** switch affects the meaning of this switch.

--objective

Change the behavior of the **--delta-value** switch so that a record is considered part of a group if the value of its **--delta-field** is within the *DELTA_VALUE* of the **first** record in the group. (When this switch is not specified, consecutive records are compared.)

--summarize

Cause **rwgroup** to print (typically) a single record for each group. By default, all records in each group having at least **--rec-threshold** members is printed. When **--summarize** is active, the record that is written for the group is the first record in the group with the following modifications:

- The packets and bytes values are the sum of the packets and bytes values, respectively, for all records in the group.
- The start-time value is the earliest start time for the records in the group.
- The end-time value is the latest end time for the records in the group.
- The flags and session-flags values are the bitwise-OR of all flags and session-flags values, respectively, for the records in the group.

Note that multiple records for a group may be printed if the bytes, packets, or elapsed time values are too large to be stored in a SiLK flow record.

--plugin=*PLUGIN*

Augment the list of fields by using run-time loading of the plug-in (shared object) whose path is *PLUGIN*. The switch may be repeated to load multiple plug-ins. The creation of plug-ins is described in the **silk-plugin(3)** manual page. When *PLUGIN* does not contain a slash (`/`), **rwgroup** will attempt to find a file named *PLUGIN* in the directories listed in the FILES section. If **rwgroup** finds the file, it uses that path. If *PLUGIN* contains a slash or if **rwgroup** does not find the file, **rwgroup** relies on your operating system's **dlopen(3)** call to find the file. When the SILK_PLUGIN_DEBUG environment variable is non-empty, **rwgroup** prints status messages to the standard error as it attempts to find and open each of its plug-ins.

--rec-threshold=*THRESHOLD*

Specify the minimum number of SiLK records a group must contain before the records in the group are written to the output stream. The default is 1; i.e., write all records. The maximum threshold is 65535.

--group-offset=*IP*

Specify the value to write into the Next Hop IP for the records that comprise the first group. The value *IP* may be an integer, or an IPv4 or IPv6 address in the canonical presentation form. If not specified, counting begins at 0. The value for each subsequent group is incremented by 1.

--note-add=*TEXT*

Add the specified *TEXT* to the header of the output file as an annotation. This switch may be repeated to add multiple annotations to a file. To view the annotations, use the **rwfileinfo(1)** tool.

--note-file-add=*FILENAME*

Open *FILENAME* and add the contents of that file to the header of the output file as an annotation. This switch may be repeated to add multiple annotations. Currently the application makes no effort to ensure that *FILENAME* contains text; be careful that you do not attempt to add a SiLK data file as an annotation.

--copy-input=*PATH*

Copy all binary SiLK Flow records read as input to the specified file or named pipe. *PATH* may be `stdout` or `-` to write flows to the standard output as long as the **--output-path** switch is specified to redirect **rwgroup**'s output to a different location.

--output-path=*PATH*

Write the binary SiLK Flow records to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output. If *PATH* names an existing file, **rwgroup** exits with an error unless the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. If this switch is not given, the output is written to the standard output. Attempting to write the binary output to a terminal causes **rwgroup** to exit with an error.

--compression-method=*COMP_METHOD*

Specify the compression library to use when writing output files. If this switch is not given, the value in the SILK_COMPRESSION_METHOD environment variable is used if the value names an available compression method. When no compression method is specified, output to the standard output or to named pipes is not compressed, and output to files is compressed using the default chosen when SiLK was compiled. The valid values for *COMP_METHOD* are determined by which external libraries were found when SiLK was compiled. To see the available compression methods and the default method, use the **--help** or **--version** switch. SiLK can support the following *COMP_METHOD* values when the required libraries are available.

none

Do not compress the output using an external library.

zlib

Use the **zlib(3)** library for compressing the output, and always compress the output regardless of the destination. Using zlib produces the smallest output files at the cost of speed.

lzo1x

Use the *lzo1x* algorithm from the LZO real time compression library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead.

snappy

Use the *snappy* library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead. *Since SiLK 3.13.0.*

best

Use lzo1x if available, otherwise use snappy if available, otherwise use zlib if available. Only compress the output when writing to a file.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwgroup** searches for the site configuration file in the locations specified in the FILES section.

--help

Print the available options and exit. Specifying switches that add new fields or additional switches before **--help** will allow the output to include descriptions of those fields or switches.

--help-fields

Print the description and alias(es) of each field and exit. Specifying switches that add new fields before **--help-fields** will allow the output to include descriptions of those fields.

--version

Print the version number and information about how SiLK was configured, then exit the application.

--pmap-file=*PATH*

--pmap-file=*MAPNAME*:*PATH*

Load the prefix map file located at *PATH* and create fields named src-*map-name* and dst-*map-name* where *map-name* is either the *MAPNAME* part of the argument or the map-name specified when the file was created (see **rwpmapbuild(1)**). If no map-name is available, **rwgroup** names the fields `sval` and `dval`. Specify *PATH* as `-` or `stdin` to read from the standard input. The switch may be repeated to load multiple prefix map files, but each prefix map must use a unique map-name. The **--pmap-file** switch(es) must precede the **--fields** switch. See also **pmapfilter(3)**.

--python-file=*PATH*

When the SiLK Python plug-in is used, **rwgroup** reads the Python code from the file *PATH* to define additional fields that can be used as part of the group key. This file should call **register_field()** for each field it wishes to define. For details and examples, see the **silkpython(3)** and **pysilk(3)** manual pages.

LIMITATIONS

rwgroup requires *sorted* data. The application works by comparing records in the order that the records are received (similar to the UNIX **uniq(1)** command), odd orders will produce odd groupings.

EXAMPLES

In the following example, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

As a rule of thumb, the **--id-fields** and **--delta-field** parameters should match **rwsort(1)**'s call, with **--delta-field** being the last parameter. A call to group all web traffic by queries from the same addresses (field=2) within 10 seconds (field=9) of the first query from that address will be:

```
$ rwfilter --proto=6 --dport=80 --pass=stdout                  \
  | rwsort --field=2,9                                         \
  | rwgroup --id-field=2 --delta-field=9 --delta-value=10      \
       --objective
```

ENVIRONMENT

PYTHONPATH

This environment variable is used by Python to locate modules. When **--python-file** is specified, **rwgroup** must load the Python files that comprise the PySiLK package, such as *silk/__init__.py*. If this *silk/* directory is located outside Python's normal search path (for example, in the SiLK installation tree), it may be necessary to set or modify the PYTHONPATH environment variable to include the parent directory of *silk/* so that Python can find the PySiLK module.

SILK_PYTHON_TRACEBACK

When set, Python plug-ins will output traceback information on Python errors to the standard error.

SILK_COUNTRY_CODES

This environment variable allows the user to specify the country code mapping file that **rwgroup** uses when computing the scc and dcc fields. The value may be a complete path or a file relative to the SILK_PATH. See the FILES section for standard locations of this file.

SILK_ADDRESS_TYPES

This environment variable allows the user to specify the address type mapping file that **rwgroup** uses when computing the sType and dType fields. The value may be a complete path or a file relative to the SILK_PATH. See the FILES section for standard locations of this file.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_COMPRESSION_METHOD

This environment variable is used as the value for **--compression-method** when that switch is not provided. *Since SiLK 3.13.0.*

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwgroup** may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files and plug-ins, **rwgroup** may use this environment variable. See the FILES section for details.

SILK_PLUGIN_DEBUG

When set to 1, **rwgroup** prints status messages to the standard error as it attempts to find and open each of its plug-ins. In addition, when an attempt to register a field fails, **rwgroup** prints a message specifying the additional function(s) that must be defined to register the field in **rwgroup**. Be aware that the output can be rather verbose.

FILES

${SILK_ADDRESS_TYPES}

${SILK_PATH}/share/silk/address_types.pmap

${SILK_PATH}/share/address_types.pmap

/usr/local/share/silk/address_types.pmap

/usr/local/share/address_types.pmap

Possible locations for the address types mapping file required by the sType and dType fields.

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

${SILK_COUNTRY_CODES}

${SILK_PATH}/share/silk/country_codes.pmap

${SILK_PATH}/share/country_codes.pmap

/usr/local/share/silk/country_codes.pmap

/usr/local/share/country_codes.pmap

Possible locations for the country code mapping file required by the scc and dcc fields.

${SILK_PATH}/lib64/silk/

${SILK_PATH}/lib64/

${SILK_PATH}/lib/silk/

${SILK_PATH}/lib/

/usr/local/lib64/silk/

/usr/local/lib64/

/usr/local/lib/silk/

/usr/local/lib/

Directories that **rwgroup** checks when attempting to load a plug-in.

SEE ALSO

rwfilter(1), **rwfileinfo(1)**, **rwsort(1)**, **rwuniq(1)**, **rwpmapbuild(1)**, **addrtype(3)**, **ccfilter(3)**, **pmapfilter(3)**, **pysilk(3)**, **silkpython(3)**, **silk-plugin(3)**, **sensor.conf(5)**, **uniq(1)**, **silk(7)**, **yaf(1)**, **dlopen(3)**, **zlib(3)**

rwidsquery

Invoke rwfilter to find flows matching Snort signatures

SYNOPSIS

```
rwidsquery --intype=INPUT_TYPE
       [--output-file=OUTPUT_FILE]
       [--start-date=YYYY/MM/DD[:HH] [--end-date=YYYY/MM/DD[:HH]]]
       [--year=YEAR] [--tolerance=SECONDS]
       [--config-file=CONFIG_FILE]
       [--mask=PREDICATE_LIST]
       [--verbose] [--dry-run]
       [INPUT_FILE | -]
       [-- EXTRA_RWFILTER_ARGS...]

 rwidsquery --help

 rwidsquery --version
```

DESCRIPTION

rwidsquery facilitates selection of SiLK flow records that correspond to Snort IDS alerts and signatures. **rwidsquery** takes as input either a **snort(8)** alert log or rule file, analyzes the alert or rule contents, and invokes **rwfilter(1)** with the appropriate arguments to retrieve flow records that match attributes of the input file. **rwidsquery** will process the Snort rules or alerts from a single file named on the command line; if no file name is given, **rwidsquery** will attempt to read the Snort rules or alerts from the standard input, unless the standard input is connected to a terminal. An input file name of `-` or `stdin` will force **rwidsquery** to read from the standard input, even when the standard input is a terminal.

OPTIONS

In addition to the options listed below, you can pass extra options through to **rwfilter(1)** on the **rwidsquery** command line. The syntax for doing so is to place a double-hyphen (--) sequence after all valid **rwidsquery** options, and before all of the options you wish to pass through to **rwfilter**.

--intype=*INPUT_TYPE*

Specify the type of input contained in the input file. This switch is required. Two alert formats and one rule format are currently supported. Valid values for this option are:

`fast`

Input is a Snort "fast" log file entry. Alerts are written in this format when Snort is configured with the `snort_fast` output module enabled. `snort_fast` alerts resemble the following:

```
Jan  1 01:23:45 hostname snort[1976]: [1:1416:11] ...
```

`full`

Input is a Snort "full" log file entry. Alerts are written in this format when Snort is configured with the `snort_full` output module enabled. `snort_full` alerts look like the following example:

```
[**] [116:151:1] (snort decoder) Bad Traffic  ...
```

`rule`

Input is a Snort rule (signature). For example:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any ...
```

--output-file=*OUTPUT_FILE*

Specify the output file that flows will be written to. If not specified, the default is to write to stdout. The argument to this option becomes the argument to **rwfilter**'s **--pass-destination** switch.

--start-date=*YYYY/MM/DD[:HH]*

--end-date=*YYYY/MM/DD[:HH]*

Used in conjunction with rule file input only. The date predicates indicate which time to start and end the search. See the **rwfilter(1)** manual page for details of the date format.

--year=*YEAR*

Used in conjunction with alert file input only. Timestamps in Snort alert files do not contain year information. By default, the current calendar year is used, but this option can be used to override this default behavior.

--tolerance=*SECONDS*

Used in conjunction with alert file input only. This option is provided to compensate for timing differences between the timestamps in Snort alerts and the start/end time of the corresponding flows. The default **--tolerance** value is 3600 seconds, which means that flow records +/- one hour from the alert timestamp will be searched.

--config-file=*CONFIG_FILE*

Used in conjunction with rule file input only. Snort requires a configuration file which, among other things, contains variables that can be used in Snort rule definitions. This option allows you to specify the location of this configuration file so that IP addresses, port numbers, and other information from the snort configuration file can be used to find matching flows.

--mask=*PREDICATE_LIST*

Exclude the **rwfilter** predicates named in *PREDICATE_LIST* from the selection criteria. This option is provided to widen the scope of queries by making them more general than the Snort rule or alert provided. For instance, **--mask**=*dport* will return flows with any destination port, not just those which match the input Snort alert or rule.

--verbose

Print the resulting **rwfilter(1)** command to the standard error prior to executing it.

--dry-run

Print the resulting **rwfilter(1)** command to the standard error but do not execute it.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

To find SiLK flows matching a Snort alert in snort_fast format:

```
$ rwidsquery --intype fast --year 2007 --tolerance 300 alert.fast.txt
```

For the following Snort alert:

```
Nov  15 00:00:58 hostname snort[5214]: [1:1416:11]
SNMP broadcast trap [Classification: Attempted Information Leak]
[Priority: 2]: {TCP}
192.168.0.1:4161 -> 127.0.0.1:139
```

The resulting **rwfilter(1)** command would look similar to:

```
$ rwfilter --start-date=2007/11/14:23 --end-date=2007/11/15:00     \
       --stime=2007/11/14:23:55:58-2007/11/15:00:05:58             \
       --saddress=192.168.0.1 --sport=4161 --daddress=127.0.0.1    \
       --dport=139 --protocol=6 --pass=stdout
```

If you want to find flows matching the same criteria, except you want UDP flows instead of TCP flows, use the following syntax:

```
$ rwidsquery --intype fast --year 2007 --tolerance 300     \
       --mask protocol alert.fast.txt -- --protocol=17
```

which would yield the following **rwfilter** command line:

```
$ rwfilter --start-date=2007/11/14:23 --end-date=2007/11/15:00     \
       --stime=2007/11/14:23:55:58-2007/11/15:00:05:58             \
       --saddress=192.168.0.1 --sport=4161 --daddress=127.0.0.1    \
       --dport=139 --protocol=17 --pass=stdout
```

To find SiLK flows matching a Snort rule:

```
$ rwidsquery --intype rule --start 2008/02/20:00 --end 2008/02/20:02 \
       --config /opt/local/etc/snort/snort.conf --verbose rule.txt
```

For the following Snort rule:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any
(msg:"ICMP Parameter Problem Bad Length"; icode:2; itype:12;
classtype:misc-activity; sid:425; rev:6;)
```

The resulting **rwfilter(1)** command would look similar to:

```
$ rwfilter --start-date=2008/02/20:00 --end-date=2008/02/20:02     \
       --stime=2008/02/20:00-2008/02/20:02                         \
       --sipset=/tmp/tmpeKIPn2.set --icmp-code=2 --icmp-type=12    \
       --pass=stdout
```

ENVIRONMENT

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_CONFIG_FILE

This environment variable is used as the location for the site configuration file, *silk.conf*. When this environment variable is not set, **rwfilter** searches for the site configuration file in the locations specified in the FILES section.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository for **rwfilter**. This value overrides the compiled-in value. In addition, **rwfilter** may use this value when searching for the SiLK site configuration files. See the FILES section for details.

SILK_RWFILTER_THREADS

The number of threads **rwfilter** uses when reading files from the data store.

SILK_PATH

This environment variable gives the root of the install tree. When searching for the site configuration file, **rwfilter** may use this environment variable. See the FILES section for details.

RWFILTER

Complete path to the **rwfilter** program. If not set, **rwidsquery** attempts to find **rwfilter** on your PATH.

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file---for report types that use **rwfilter**.

SEE ALSO

rwfilter(1), **silk(7)**, **snort(8)**

rwip2cc

Maps IP addresses to country codes

SYNOPSIS

```
rwip2cc { --address=IP_ADDRESS | --input-file=FILE }
      [--map-file=PMAP_FILE] [--print-ips={0,1}]
      [{--integer-ips | --zero-pad-ips}] [--no-columns]
      [--column-separator=CHAR] [--no-final-delimiter]
      [{--delimited | --delimited=CHAR}]
      [--output-path=PATH] [--pager=PAGER_PROG]

rwip2cc --help

rwip2cc --version
```

DESCRIPTION

As of SiLK 3.0, **rwip2cc** is deprecated, and it will be removed in the SiLK 4.0 release. Use **rwpmaplookup(1)** instead---the EXAMPLES section shows how to use **rwpmaplookup** to get output similar to that produced by **rwip2cc**.

rwip2cc maps from (textual) IP address to two letter country code. Either the **--address** or **--input-file** switch is required.

The **--address** switch looks up the country code of a single IP address and prints the country code to the standard output.

The **--input-file** switch reads data from the specified file (use `stdin` or `-` to read from the standard input) and prints, to the standard output, the country code for each IP it sees. Blank lines in the input are ignored; comments, which begin at the `#` character and extend to the end of line, are also ignored. Each line that is not a blank or a comment should contain an IP address or a CIDR block; **rwip2cc** will complain if the line cannot be parsed. Note that for CIDR blocks, the CIDR block is exploded into its constituent IP addresses and the country code for **each** IP address is printed.

The **--print-ips** switch controls whether the IP is printed with its country code. When **--print-ips=1** is specified, the output contains two columns: the IP and the country-code. When **--print-ips=0** is specified, only the country code is given. The default behavior is to print the IP whenever the **--input-file** switch is provided, and not print the IP when **--address** is given.

You can tell **rwip2cc** to use a specific country code prefix map file by giving the location of that file to the **--map-file** switch. The country code prefix map file is created with the **rwgeoip2ccmap(1)** command. When **--map-file** is not specified, **rwip2cc** attempts to use the default country code mapping file, as specified in the FILES section below.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--address=*IP_ADDRESS*

Print to the standard output the country code for the single *IP_ADDRESS*.

--input-file=*FILE*

Print the IP and country code for each IP address in *FILE*; use `stdin` to read from the standard input.

--map-file=*PMAP_FILE*

Use the designated country code prefix mapping file instead of the default.

--print-ips={0|1}

Controls whether the IP is printed. When the value is 1, the output contains two columns: the IP and the country-code. When the value is 0, only the country code is given. When this switch is not specified, the default behavior is to print the IPs only when input comes from a file (i.e., when **--input-file** is specified).

--integer-ips

Enable printing of IPs and print the IPs as integers. By default, IP addresses are printed in their canonical form.

--zero-pad-ips

Enable printing of IPs and print the IP addresses in their canonical form, but add zeros to the IP address so it fully fills the width of column. For IPv4, use three digits per octet, e.g, `127.000.000.001`.

--no-columns

Disable fixed-width columnar output.

--column-separator=*C*

Use specified character between columns and after the final column. When this switch is not specified, the default of '|' is used.

--no-final-delimiter

Do not print the column separator after the final column. Normally a delimiter is printed.

--delimited

--delimited=*C*

Run as if **--no-columns --no-final-delimiter --column-sep**=*C* had been specified. That is, disable fixed-width columnar output; if character *C* is provided, it is used as the delimiter between columns instead of the default '|'.

--output-path=*PATH*

Write the textual output to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output (and bypass the paging program). If *PATH* names an existing file, **rwip2cc** exits with an error unless the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. If this option is not given, the output is either sent to the pager or written to the standard output.

--pager=*PAGER_PROG*

When the **--input-file** switch is specified and output is to a terminal, invoke the program *PAGER_PROG* to view the output one screen full at a time. This switch overrides the SILK_PAGER environment variable, which in turn overrides the PAGER variable. If the **--output-path** switch is given or if the value of the pager is determined to be the empty string, no paging is performed and all output is written to the terminal.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

The following examples demonstrate the use of **rwip2cc**. In addition, each example shows how to get similar output using **rwpmaplookup(1)**.

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

Single address specified on the command line

Print the country code for a single address using the default country code map. By default, only the value is printed when the address is specified on the command line.

```
$ rwip2cc --address=10.0.0.0
--
```

Use the **--print-ips** switch to print the address and the country.

```
$ rwip2cc --print-ip=1 --address=10.0.0.0
       10.0.0.0|--|
```

rwpmaplookup expects the input to come from a file, so use the **--no-files** switch to tell **rwpmaplookup** that the command line arguments are the addresses to print. By default, **rwpmaplookup** prints a title line, and each row contains the key and the value.

```
$ rwpmaplookup --country-code --no-files 10.0.0.0
          key|value|
     10.0.0.0|   --|
```

Use **rwpmaplookup**'s command line switches to exactly mimic the default output from **rwip2cc**:

```
$ rwpmaplookup --country-code --fields=value --delimited --no-title \
    --no-files 10.0.0.0
--
```

Single address using a different country code file

Print the country code for a single address specified on the command line using an older version of the country code mapping file.

```
$ rwip2cc --map-file=old-addresses.pmap --address=128.2.0.0
us
```

```
$ rwpmaplookup --country-code=old-address-map.pmap --no-files 128.2.0.0
          key|value|
    128.2.0.0|   us|
```

Addresses read from the standard input

Using the default country code map, print the country code for multiple addresses read from the standard input. When the **--input-file** switch is given, the default output includes the address.

```
$ echo '10.0.0.0/31' | rwip2cc --input-file=stdin
       10.0.0.0|--|
       10.0.0.1|--|
```

You can use the **--print-ips** switch to suppress the IPs.

```
$ echo '10.0.0.0/31' | rwip2cc --print-ips=0 --input-file=stdin
--
--
```

Unlike **rwip2cc**, **rwpmaplookup** does not accept CIDR blocks as input. Use the IPset tools **rwsetbuild(1)** to parse the CIDR block list and **rwsetcat(1)** to print the list.

```
$ echo '10.0.0.0/31' | rwsetbuild | rwsetcat --cidr=0 \
    | rwpmaplookup --country-code
          key|value|
       10.0.0.0|   --|
       10.0.0.1|   --|
```

Addresses read from a file

Using an older version of the country code map, print the country code for multiple addresses read from a file.

```
$ export SILK_COUNTRY_CODES=old-addresses.pmap
$ cat file.txt
128.2.1.1
128.2.2.2
$ rwip2cc --input-file=file.txt
      128.2.1.1|us|
      128.2.2.2|us|
```

```
$ rwpmaplookup --no-title --country-code file.txt
     128.2.1.1|   us|
     128.2.2.2|   us|
```

ENVIRONMENT

SILK_COUNTRY_CODES

This environment variable allows the user to specify the country code mapping file that **rwip2cc** will use. The value may be a complete path or a file relative to SILK_PATH. If the variable is not specified, the code looks for a file named *country_codes.pmap* as specified in the FILES section below.

SILK_PATH

This environment variable gives the root of the install tree. As part of its search for the Country Code mapping file, **rwip2cc** checks the directories *$SILK_PATH/share/silk* and *$SILK_PATH/share* for a file named *country_codes.pmap*.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_PAGER

When set to a non-empty string, **rwip2cc** automatically invokes this program to display its output a screen at a time. If set to an empty string, **rwip2cc** does not automatically page its output.

PAGER

When set and SILK_PAGER is not set, **rwip2cc** automatically invokes this program to display its output a screen at a time.

FILES

rwip2cc will look for the prefix map file that maps IPs to country codes in the following locations. ($SILK_COUNTRY_CODES is the value of the SILK_COUNTRY_CODES environment variable, if it is set. $SILK_PATH is value of the SILK_PATH environment variable, if it is set. The use of */usr/local/* assumes the application is installed in the */usr/local/bin/* directory.)

```
$SILK_COUNTRY_CODES
$SILK_PATH/share/silk/country_codes.pmap
$SILK_PATH/share/country_codes.pmap
/usr/local/share/silk/country_codes.pmap
/usr/local/share/country_codes.pmap
```

SEE ALSO

rwpmaplookup(1), **rwgeoip2ccmap(1)**, **rwsetbuild(1)**, **rwsetcat(1)**, **silk(7)**

rwipaexport

Export IPA datasets to SiLK binary data files

SYNOPSIS

```
rwipaexport --catalog=CATALOG [--time=TIME] [--prefix-map-name=NAME]
      [--note-add=TEXT] [--note-file-add=FILE]
      [--compression-method=COMP_METHOD] OUTPUT_FILE


rwipaexport --help


rwipaexport --version
```

DESCRIPTION

rwipaexport exports data from an IPA (IP Association, http://tools.netsa.cert.org/ipa/) data store to a SiLK IPset, Bag, or prefix map file, depending on the type of the stored IPA catalog. For catalogs with time information (e.g. time period at which the stored data is considered valid) data can be selected for a specific time of interest.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--catalog=*CATALOG_NAME*

Specifies the name of the IPA catalog to export from.

--time=*TIME*

This argument allows you to export a dataset that was active at *TIME*. The expected format of this option is YYYY/MM/DD[:HH[:MM[:SS]]]. A dataset will only be returned if *TIME* falls between the start and end time for the dataset. If this option is not specified, the current time will be used. See the **TIME RANGES** section of **ipaimport(1)** for more information about how time ranges are used in IPA.

--prefix-map-name=*NAME*

When creating a prefix map file, add *NAME* to the header of the file as the map-name. When this switch is not specified, no map-name is written to the file. If the output is not a prefix map file, the **--prefix-map-file** switch is ignored.

--note-add=*TEXT*

Add the specified *TEXT* to the header of the output file as an annotation. This switch may be repeated to add multiple annotations to a file. To view the annotations, use the **rwfileinfo(1)** tool.

--note-file-add=*FILENAME*

Open *FILENAME* and add the contents of that file to the header of the output file as an annotation. This switch may be repeated to add multiple annotations. Currently the application makes no effort to ensure that *FILENAME* contains text; be careful that you do not attempt to add a SiLK data file as an annotation.

--compression-method=*COMP_METHOD*

Specify the compression library to use when writing output files. If this switch is not given, the value in the SILK_COMPRESSION_METHOD environment variable is used if the value names an available compression method. When no compression method is specified, output to the standard output or to named pipes is not compressed, and output to files is compressed using the default chosen when SiLK was compiled. The valid values for *COMP_METHOD* are determined by which external libraries were found when SiLK was compiled. To see the available compression methods and the default method, use the **--help** or **--version** switch. SiLK can support the following *COMP_METHOD* values when the required libraries are available.

none

Do not compress the output using an external library.

zlib

Use the **zlib(3)** library for compressing the output, and always compress the output regardless of the destination. Using zlib produces the smallest output files at the cost of speed.

lzo1x

Use the *lzo1x* algorithm from the LZO real time compression library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead.

snappy

Use the *snappy* library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead. *Since SiLK 3.13.0.*

best

Use lzo1x if available, otherwise use snappy if available, otherwise use zlib if available. Only compress the output when writing to a file.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

To export the `badhosts` IPset from an IPA set catalog into the file *badhosts.set* where there is no time information:

```
$ rwipaexport --catalog=badhosts badhosts.set
```

To export the `flowcount` Bag from an IPA bag catalog into the file *flowcount-20070415.bag* where there is time information:

```
$ rwipaexport --catalog=flowcount --time=2007/04/15  \
       flowcount-20070415.bag
```

ENVIRONMENT

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_COMPRESSION_METHOD

This environment variable is used as the value for **--compression-method** when that switch is not provided. *Since SiLK 3.13.0.*

SILK_PATH

This environment variable gives the root of the directory tree where the tools are installed. When searching for the *silk-ipa.conf* configuration file, **rwipaexport** may use this environment variable. See the FILES section for details.

FILES

$SILK_PATH/share/silk/silk-ipa.conf

$SILK_PATH/share/silk-ipa.conf

/usr/local/share/silk/silk-ipa.conf

/usr/local/share/silk-ipa.conf

Possible locations for the IPA configuration file. This file contains the URI for connecting to the IPA database. If the configuration file does not exist, **rwipaexport** will exit with an error. The format of this URI is *driver*://*user*:*pass-word*@*hostname*/*database*. For example:

```
postgresql://ipauser:secret@database-server.domain.com/ipa
```

SEE ALSO

rwipaimport(1), **rwfileinfo(1)**, **ipafilter(3)**, **silk(7)**, **ipaimport(1)**, **ipaexport(1)**, **ipaquery(1)**, **zlib(3)**

rwipaimport

Import SiLK IP collections into an IPA catalog

SYNOPSIS

```
rwipaimport --catalog=CATALOG [--description=DESCRIPTION]
      [--start-time=START_TIME] [--end-time=END_TIME] INPUT_FILE

rwipaimport --help

rwipaimport --version
```

DESCRIPTION

rwipaimport reads a SiLK IPset, Bag, or Prefix Map file and imports its contents into an IPA (IP Association, http://tools.netsa.cert.org/ipa/) catalog. An IPA catalog is a collection of sets, bags, and prefix maps which can have an optional time period associated with them defining when that particular collection of data is considered valid.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--catalog=*CATALOG_NAME*

Specifies the name of the IPA catalog to import into. If the catalog does not already exist in the IPA data store, it will be created. This option is required.

--description=*DESCRIPTION*

An optional text description of the catalog's contents. This description will be stored in the database and will be visible when querying available catalogs with the **ipaquery** tool. The description will only be added to new catalogs; if you import a dataset into an existing catalog, this option is ignored.

--start-time=*START_TIME*

Specifies the beginning of the time range for which the imported data is valid. The expected format of this option is either a timestamp in YYYY/MM/DD[:HH[:MM[:SS]]] format, or `...` (three dots) to indicate the time range is left-unbounded. For more information about this argument, refer to the **TIME RANGES** section of **ipaimport(1)**.

--end-time=*END_TIME*

Specifies the end of the time range for which the imported data is valid. The expected format of this option is either a timestamp in YYYY/MM/DD[:HH[:MM[:SS]]] format, or `...` (three dots) to indicate the time range is right-unbounded. For more information about this argument, refer to the **TIME RANGES** section of **ipaimport(1)**.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

To import the IPset file *test-april.set* into a new catalog with the name `testset` and a short description, with data valid for only the month of April, 2007:

```
$ rwipaimport --catalog=testset --desc="Test set catalog"      \
        --start=2007/04/01 --end=2007/05/01                    \
        test-april.set
```

To import the Bag file *test.bag* into a new catalog named `testbag` with data valid for all dates and times (the `...` literally means the characters `...`):

```
$ rwipaimport --catalog=testbag --start=... --end=... test.bag
```

ENVIRONMENT

SILK_PATH

This environment variable gives the root of the directory tree where the tools are installed. When searching for the *silk-ipa.conf* configuration file, **rwipaimport** may use this environment variable. See the FILES section for details.

FILES

$SILK_PATH/share/silk/silk-ipa.conf

$SILK_PATH/share/silk-ipa.conf

/usr/local/share/silk/silk-ipa.conf

/usr/local/share/silk-ipa.conf

Possible locations for the IPA configuration file. This file contains the URI for connecting to the IPA database. If the configuration file does not exist, **rwipaimport** will exit with an error. The format of this URI is *driver*://*user*:*pass-word*@*hostname*/*database*. For example:

```
postgresql://ipauser:secret@database-server.domain.com/ipa
```

SEE ALSO

rwipaexport(1), **ipafilter(3)**, **silk(7)**, **ipaimport(1)**, **ipaexport(1)**, **ipaquery(1)**

rwipfix2silk

Convert IPFIX records to SiLK Flow records

SYNOPSIS

```
rwipfix2silk [--silk-output=PATH] [--print-statistics]
      [--interface-values={snmp | vlan}]
      [--log-destination={stdout | stderr | none | PATH}]
      [--log-flags=FLAGS] [--note-add=TEXT] [--note-file-add=FILE]
      [--compression-method=COMP_METHOD]
      {[--xargs] | [--xargs=FILENAME] | [IPFIXFILE [IPFIXFILE...]]}

rwipfix2silk --help

rwipfix2silk --version
```

DESCRIPTION

rwipfix2silk reads IPFIX (Internet Protocol Flow Information eXport) records from files or from the standard input, converts the records to the SiLK Flow format, and writes the SiLK records to the path specified by **--silk-output** or to the standard output when stdout is not the terminal and **--silk-output** is not provided.

rwipfix2silk reads IPFIX records from the files named on the command line or from the standard input when no file names are specified and **--xargs** is not present. To read the standard input in addition to the named files, use `-` or `stdin` as a file name. When the **--xargs** switch is provided, **rwipfix2silk** reads the names of the files to process from the named text file or from the standard input if no file name argument is provided to the switch. The input to **--xargs** must contain one file name per line.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--silk-output=*PATH*

Write the binary SiLK Flow records to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output. If *PATH* names an existing file, **rwipfix2silk** exits with an error unless the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. If this switch is not given, the output is written to the standard output. Attempting to write the binary output to a terminal causes **rwipfix2silk** to exit with an error.

--print-statistics

Print, to the standard error, the number of records that were written to the SiLK output file. See also **--log-destination**.

--interface-values={snmp | vlan}

Specify which IPFIX fields should be stored in the `input` and `output` fields of the generated SiLK Flow records. If this switch is not specified, the default is **snmp**. The choices are:

snmp

Store the indexes of the network interface cards where the flows entered and left the router. That is, store the `ingressInterface` in `input` and the `egressInterface` in `output`.

vlan

Store the VLAN identifiers for the source and destination networks. That is, store `vlanId` in `input` and `postVlanId` in `output`. If only one VLAN ID is available, `input` is set to that value and `output` is set to 0.

--log-destination={none | stdout | stderr | *PATH*}

Write more detailed information to the specified destination. The default destination is **none** which suppresses messages. Use **stdout** or **stderr** to send messages to the standard output or standard error, respectively. Any other value is treated as a file name in which to write the messages. When an existing file is specified, **rwipfix2silk** appends any messages to the file. Information that is written includes the following:

- For each input stream, the number of forward and reverse IPFIX records read and number of records ignored.
- Messages about invalid records.
- When the SILK_IPFIX_PRINT_TEMPLATES environment variable is set to 1, the IPFIX templates that were read.
- Additional messages enabled by the **--log-flags** switch.

--log-flags=*FLAGS*

Write additional messages regarding the IPFIX data to the **--log-destination**, where *FLAGS* is a comma-separated list of names specifying the type messages to write. When this switch is not specified, the default value for *FLAGS* is `none`. This switch takes the same values as the **log-flags** setting in the **sensor.conf(5)** file. This manual page documents the values that are relevant for IPFIX data. *Since SiLK 3.10.2.*

all

Log everything.

default

Enable the default set of log-flags used by *sensor.conf*: **sampling**. Despite the name, this is not the default setting for this switch; **none** is.

none

Log nothing. It is an error to combine this log-flag name with any other. This is the default setting for **--log-flags**.

record-timestamps

Log the timestamps that appear on each record. This produces a lot of output, and it is primarily used for debugging.

sampling

Write messages constructed by parsing the IPFIX Options Templates that specify the sampling algorithm (when samplingAlgorithm and samplingInterval IEs are present) or flow sampler mode (when flowSamplerMode and flowSamplerRandomInterval IEs are present).

show-templates

Write messages to the log describing each IPFIX template that is read. The message contains embedded new lines, with the template ID and domain on the first line, and each of the template's elements on the following lines. Each element is described by its name, its IE number with the private enterprise number if any, and its length in the template. The format is that described in Section 10.2 of RFC7013. *Since SiLK 3.19.0.*

--note-add=*TEXT*

Add the specified *TEXT* to the header of the output file as an annotation. This switch may be repeated to add multiple annotations to a file. To view the annotations, use the **rwfileinfo(1)** tool.

--note-file-add=*FILENAME*

Open *FILENAME* and add the contents of that file to the header of the output file as an annotation. This switch may be repeated to add multiple annotations. Currently the application makes no effort to ensure that *FILENAME* contains text; be careful that you do not attempt to add a SiLK data file as an annotation.

--compression-method=*COMP_METHOD*

Specify the compression library to use when writing output files. If this switch is not given, the value in the SILK_COMPRESSION_METHOD environment variable is used if the value names an available compression method. When no compression method is specified, output to the standard output or to named pipes is not compressed, and output to files is compressed using the default chosen when SiLK was compiled. The valid values for *COMP_METHOD* are determined by which external libraries were found when SiLK was compiled. To see the available compression methods and the default method, use the **--help** or **--version** switch. SiLK can support the following *COMP_METHOD* values when the required libraries are available.

none

Do not compress the output using an external library.

zlib

Use the **zlib(3)** library for compressing the output, and always compress the output regardless of the destination. Using zlib produces the smallest output files at the cost of speed.

lzo1x

Use the *lzo1x* algorithm from the LZO real time compression library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead.

snappy

Use the *snappy* library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead. *Since SiLK 3.13.0.*

best

Use lzo1x if available, otherwise use snappy if available, otherwise use zlib if available. Only compress the output when writing to a file.

--xargs

--xargs=*FILENAME*

Read the names of the input files from *FILENAME* or from the standard input if *FILENAME* is not provided. The input is expected to have one filename per line. **rwipfix2silk** opens each named file in turn and reads records from it as if the filenames had been listed on the command line.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

To convert a packet capture (**pcap(3)**) file, *packets.pcap*, such as that produced by **tcpdump(1)**, to the SiLK format, use the **yaf(1)** tool (see http://tools.netsa.cert.org/yaf/) to convert the capture data to IPFIX and **rwipfix2silk** to convert the IPFIX data to the SiLK format, storing the records in *silk.rw*:

```
$ yaf --silk --in packets.pcap --out -              \
  | rwipfix2silk --silk-output=silk.rw
```

Note that you can produce the same result using the **rwp2yaf2silk(1)** wrapper script:

```
$ rwp2yaf2silk --in packets.pcap --out silk.rw
```

You can use **rwsilk2ipfix(1)** to convert the SiLK file back to an IPFIX format, storing the result in *ipfix.dat*:

```
$ rwsilk2ipfix --silk-output=silk.rw ipfix.dat
```

If you want to create flow records that contain a single packet (similar to the output of **rwptoflow(1)**), specify **--idle-timeout=0** on the **yaf** command line:

```
$ yaf --silk --in packets.pcap --out - --idle-timeout=0    \
  | rwipfix2silk --silk-output=silk.rw
```

To have **yaf** decode VLAN identifiers for 802.1Q packets and to have **rwipfix2silk** store the VLAN IDs in the `input` and `output` fields of the SiLK Flow records, use:

```
$ yaf --silk --in packets.pcap --out -                              \
  | rwipfix2silk --silk-output=silk.rw --interface-values=vlan
```

Note: **yaf** releases prior to 1.3 would only export the VLAN identifiers when the **--mac** switch was provided on the command line.

ENVIRONMENT

SILK_IPFIX_PRINT_TEMPLATES

When set to 1, **rwipfix2silk** adds `show-templates` to the **--log-flags** switch. See the description of that switch for additional information.

SILK_LIBFIXBUF_SUPPRESS_WARNINGS

When set to 1, **rwipfix2silk** disables all warning messages generated by libfixbuf. These warning messages include out-of-sequence packets, data records not having a corresponding template, record count discrepancies, and issues decoding list elements. *Since SiLK 3.10.0.*

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_COMPRESSION_METHOD

This environment variable is used as the value for **--compression-method** when that switch is not provided. *Since SiLK 3.13.0.*

SEE ALSO

rwsilk2ipfix(1), **rwfileinfo(1)**, **rwp2yaf2silk(1)**, **rwptoflow(1)**, **sensor.conf(5)**, **silk(7)**, **yaf(1)**, **tcpdump(1)**, **pcap(3)**, **zlib(3)**

rwmatch

Match SiLK records from two streams into a common stream

SYNOPSIS

```
rwmatch --relate=FIELD_PAIR [--relate=FIELD_PAIR ...]
      [--time-delta=DELTA] [--symmetric-delta]
      [{ --absolute-delta | --relative-delta | --infinite-delta }]
      [--unmatched={q|r|b}]
      [--note-add=TEXT] [--note-file-add=FILE]
      [--ipv6-policy={ignore,asv4,mix,force,only}]
      [--compression-method=COMP_METHOD]
      [--site-config-file=FILENAME]
      QUERY_FILE RESPONSE_FILE OUTPUT_FILE

rwmatch --help

rwmatch --help-relate

rwmatch --version
```

DESCRIPTION

rwmatch provides a facility for relating (or matching) SiLK Flow records contained in two **sorted** input files, labeling those flow records, and writing the records to an output file.

The two input files are called *QUERY_FILE* and *RESPONSE_FILE*, respectively. The purpose of **rwmatch** is to find a record in *QUERY_FILE* that represents some network stimulus that caused a reply which is represented by a record in *RESPONSE_FILE*. When **rwmatch** discovers this relationship, it assigns a numeric ID to the match, searches both input files for additional records that are part of the same event, stores the numeric ID in each matching record's next hop IP field, and writes all records that are part of that event to *OUTPUT_FILE*.

When the **--symmetric-delta** switch is specified, **rwmatch** also checks for a stimulus in *RESPONSE_FILE* that triggered a reply in *QUERY_FILE*. This is useful when matching flows where either side may have initiated the conversation.

The input files must be sorted as described in Sorting the input below. To use the standard input in place of one of the input streams, specify `stdin` or `-` in its place.

The criteria for defining a match are given by one of more uses of the **--relate** switch and by the timestamps on the flow records:

- Each use of **--relate** on the command line takes two comma-separated SiLK Flow record fields as its argument. These two fields form a *FIELD_PAIR* in the form *QUERY_FIELD*,*RESPONSE_FIELD*. For a match to exist, the value of *QUERY_FIELD* on a record read from *QUERY_FILE* must be identical to the value of *RESPONSE_FIELD* on a record read from *RESPONSE_FILE*, and that must be true for all *FIELD_PAIR*s.

- By default, the start-time of the record from the *RESPONSE_FILE* must begin within a time window determined by the start- and end-times of the record read from the *QUERY_FILE*. The end-time is extended by specifying the *DELTA* number of seconds as the argument to the **--time-delta** switch. Thus

  ```
  query_rec.sTime <= response_rec.sTime <= query_rec.eTime + DELTA
  ```

  When the **--symmetric-delta** switch is provided, records also match if the start-time of the query record begins within the time window determined by the start- and end-times of the response record, plus any value specified by **--time-delta**. That is:

  ```
  response_rec.sTime <= query_rec.sTime <= response_rec.eTime + DELTA
  ```

  The **--time-delta** switch allows for a delay in the response. Although responses usually occur within a second of the query, delays of several seconds are not uncommon due to combinations of host and network processing delays. The *DELTA* value can also compensate for timing errors between multiple sensors.

Once **rwmatch** establishes a match between records in the two input files, it searches for additional records from both input files to add to the match.

To do this, **rwmatch** denotes one of the records that comprise the initial match pair as a *base* record. When possible, the base record is the record with the earlier start time. In the case of a tie, the base is determined by ports for TCP and UDP with the base being that with the lower port if one is above 1024 and the other below 1024. If that also fails, the base record is the record read from *QUERY_FILE*. With millisecond time resolution, ties should be rare.

To determine whether a match exists between the base record and a *candidate* record, **rwmatch** uses the *FIELD_PAIR*s specified by **--relate**. When the base record and the candidate record were read from the same file, only one side of each *FIELD_PAIR* is used.

In addition to the records having identical values for each field in *FIELD_PAIR*s, the candidate record must be within a time window determined by the **--time-delta** switch and the **--absolute-delta**, **--relative-delta**, and **--infinite-delta** switches.

- When **--infinite-delta** is specified, there is no time window and only the values specified by the *FIELD_PAIR*s are checked.
- Specifying **--absolute-delta** requires each candidate record to start within the time window set by the start- and end-times of the base record (plus any *DELTA*), similar to the rule used to establish the match.
- If **--relative-delta** is specified, the end of the time window is initially set to *DELTA* seconds after the end-time of the base record. As records from either input file are added to the match, the end of the time window is set to *DELTA* seconds beyond the maximum end-time seen on any record in the match.
- When none of the above are explicitly specified, **rwmatch** uses the rules of **--absolute-delta**.

Because long-lived sessions are often broken into multiple flows, **rwmatch** may discard records that are part of a long-lived session. The **--relative-delta** switch may compensate for this if the gap between flows is less that the time specified in the **--time-delta** switch. The **--infinite-delta** will compensate for arbitrarily long gaps, but it may add records to a match that are not part of a true session. DNS flows that use port 53/udp as both a service and reply port are an example.

When **rwmatch** establishes a match, it increments the match ID, with the first match having a match ID of 1. To label the records that comprise the match, **rwmatch** uses a 32-bit number where the lower 24-bits hold the match ID and the upper 8-bits is set to 0 or 255 to indicate whether the record was read from *QUERY_FILE* or *RESPONSE_FILE*, respectively. **rwmatch** stores this 32-bit number in the next hop IP field of the records. If the record is IPv6, **rwmatch** maps the number into the ::ffff:0:0/96 netblock before modifying setting the next hop IP. Apart from the change to the next hop IP field, the query and response records are not modified.

By default, only matched records are written to the *OUTPUT_FILE* and any record that could not be determined to be part of a match is discarded.

Specifying the **--unmatched** switch tells **rwmatch** to write unmatched query and/or response records to *OUTPUT_FILE*. The required parameter is one of `q`, `r`, or `b` to write the query records, the response records, or both to *OUTPUT_FILE*. Unmatched query records have their next hop IP set to 0.0.0.0, and unmatched response records have their next hop IP set to 255.0.0.0.

Sorting the input

As **rwmatch** reads *QUERY_FILE* and *RESPONSE_FILE*, it expects the SiLK Flow records to appear in a particular order that is best achieved by using **rwsort(1)**. In particular:

- The records in *QUERY_FILE* must appear in ascending order where the key is the first value in each of the **--relate** *FIELD_PAIR*s in the order in which the **--relate** switches appear and by the start time of the flow.
- Likewise for the records in *RESPONSE_FILE*, except the second value in each *FIELD_PAIR*s is used.

When **rwmatch** processes the following command

```
$ rwmatch --relate=1,2 --relate=2,1 --relate=5,5 Q.rw R.rw out.rw
```

it assumes the *file1.rw* and *file2.rw* were created by

```
$ rwsort --fields=1,2,5,stime --output=Q.rw input1.rw ....
$ rwsort --fields=2,1,5,stime --output=R.rw input2.rw ....
```

If the files *source_ips.s.rw* and *dest_ips.s.rw* are created by the following commands:

```
$ rwsort --field=1,9 source_ips.rw > source_ips.s.rw
$ rwsort --field=2,9 dest_ips.rw > dest_ips.s.rw
```

The following call to **rwmatch** works correctly:

```
$ rwmatch --relate=1,2 source_ips.s.rw dest_ips.s.rw matched.rw
```

Note that the following command produces very few matches since *source_ips.s.rw* was sorted on field 1 and *dest_ips.s.rw* was sorted on field 2.

```
$ rwmatch --relate=2,1 source_ips.s.rw dest_ips.s.rw stdout
```

The recommended sort ordering for TCP and UDP is shown below. This correctly handles multiple flows occurring during the same time interval which involve multiple ports:

```
$ rwsort --fields=1,4,2,3,5,stime incoming.rw > incoming-query.rw
$ rwsort --fields=2,3,1,4,5,stime outgoing.rw > outgoing-response.rw
```

The corresponding **rwmatch** command is:

```
$ rwmatch --relate=1,2 --relate=4,3 --relate=2,1 --relate=3,4 \
       --relate=5,5 incoming-query.rw outgoing-response.rw matched.rw
```

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--relate=*FIELD_PAIR*

Specify a pair of fields where the value of these fields in two records must be identical for the records to be considered part of a match. The first field is for records from *QUERY_FILE* and the second for records from *RESPONSE_FILE*. At least one *FIELD_PAIR* must be provided; up to 128 *FIELD_PAIR*s may be provided. The *FIELD_PAIR* must contain two field names or field IDs separated by a comma, such as **--relate=dip,sip** or **--relate=proto,proto**.

Each *FIELD_PAIR* is unidirectional; specifying **--relate=sip,dip** matches records where the query record's source IP matches the response record's destination IP, but does not imply any relationship between the response's source IP and query's destination IP. To match symmetric flow records between hosts, specify:

```
--relate=sip,dip --relate=dip,sip
```

When using a port-based protocol (e.g., TCP or UDP), refine the match further by specifying the ports:

```
--relate=2,1 --relate=1,2 --relate=3,4 --relate=4,3
```

Matching becomes more specific as more fields are added. Since **rwmatch** discards unmatched records, a highly specific match (such as the last one specified above) generates more matches (resulting in higher match IDs), but may result in fewer total flows due to certain records being unmatched.

The available fields are listed here. For a better description of some of these fields, see the **rwcut(1)** manual page.

sIP,1
: source IP address

dIP,2
: destination IP address

sPort,3
: source port for TCP and UDP, or equivalent

dPort,4
destination port for TCP and UDP, or equivalent

protocol,5
IP protocol

packets,pkts,6
packet count

bytes,7
byte count

flags,8
bit-wise OR of TCP flags over all packets

sensor,12
name or ID of sensor at the collection point

class,20
class of sensor at the collection point

type,21
type of sensor at the collection point

iType
the ICMP type value for ICMP or ICMPv6 flows and empty for non-ICMP flows. This field was introduced in SiLK 3.8.1.

iCode
the ICMP code value for ICMP or ICMPv6 flows and empty for non-ICMP flows. See note at `iType`.

in,13
router SNMP input interface or vlanId if packing tools were configured to capture it (see **sensor.conf(5)**)

out,14
router SNMP output interface or postVlanId

initialFlags,26
TCP flags on first packet in the flow

sessionFlags,27
bit-wise OR of TCP flags over all packets except the first in the flow

attributes,28
flow attributes set by the flow generator

application,29
guess as to the content of the flow

--time-delta=*DELTA*

Specify the number of seconds by which a response record may start after a query record has ended. *DELTA* may contain fractional seconds to millisecond precision; for example, 0.500 represents a 500 millisecond delay. Responses match queries if

```
query.sTime <= response.sTime <= query.eTime + DELTA
```

When **--time-delta** is not specified, *DELTA* defaults to 0 and the response must begin before the query ends.

--symmetric-delta

Allow matching of flows where the *RESPONSE_FILE* contains the initial flow. In this case, a query record matches a response record when

```
response.sTime <= query.sTime <= response.eTime + DELTA
```

--absolute-delta

When adding additional records to an established match, only include candidate flows that start less than *DELTA* seconds after the end of the initial flow. This is the default behavior. This switch is incompatible with **--relative-delta** and **--infinite-delta**.

--relative-delta

When adding additional records to an established match, include candidate flows that start within *DELTA* seconds of the greatest end time for all records in the current match. This switch is incompatible with **--absolute-delta** and **--infinite-delta**.

--infinite-delta

When adding additional records to an established match, include candidate records based on the *FIELD_PAIRS* alone, ignoring time. This switch is incompatible with **--absolute-delta** and **--relative-delta**.

--unmatched=q|r|b

Write unmatched query and/or response records to *OUTPUT_FILE*. The parameter determines whether the query records, the response records, or both are written to *OUTPUT_FILE*. Unmatched query records have their next hop IPv4 address set to 0.0.0.0, and unmatched response records have their next hop IPv4 address set to 255.0.0.0. When the **b** value is used, *OUTPUT_FILE* contains a complete merge of *QUERY_FILE* and *RESPONSE_FILE*.

--note-add=*TEXT*

Add the specified *TEXT* to the header of the output file as an annotation. This switch may be repeated to add multiple annotations to a file. To view the annotations, use the **rwfileinfo(1)** tool.

--note-file-add=*FILENAME*

Open *FILENAME* and add the contents of that file to the header of the output file as an annotation. This switch may be repeated to add multiple annotations. Currently the application makes no effort to ensure that *FILENAME* contains text; be careful that you do not attempt to add a SiLK data file as an annotation.

--ipv6-policy=*POLICY*

Determine how IPv4 and IPv6 flows are handled when SiLK has been compiled with IPv6 support. When the switch is not provided, the SILK_IPV6_POLICY environment variable is checked for a policy. If it is also unset or contains an invalid policy, the *POLICY* is **mix**. When SiLK has not been compiled with IPv6 support, IPv6 flows are always ignored, regardless of the value passed to this switch or in the SILK_IPV6_POLICY variable. The supported values for *POLICY* are:

ignore

Ignore any flow record marked as IPv6, regardless of the IP addresses it contains.

asv4

Convert IPv6 flow records that contain addresses in the ::ffff:0:0/96 netblock (that is, IPv4-mapped IPv6 addresses) to IPv4 and ignore all other IPv6 flow records.

mix

Process the input as a mixture of IPv4 and IPv6 flow records. Should **rwmatch** need to compare an IPv4 and IPv6 address, it maps the IPv4 address into the ::ffff:0:0/96 netblock.

force

Convert IPv4 flow records to IPv6, mapping the IPv4 addresses into the ::ffff:0:0/96 netblock.

only

Process only flow records that are marked as IPv6 and ignore IPv4 flow records in the input.

--compression-method=*COMP_METHOD*

Specify the compression library to use when writing output files. If this switch is not given, the value in the SILK_COMPRESSION_METHOD environment variable is used if the value names an available compression method. When no compression method is specified, output to the standard output or to named pipes is not compressed, and output to files is compressed using the default chosen when SiLK was compiled. The valid values for *COMP_METHOD* are determined by which external libraries were found when SiLK was compiled. To see the available compression methods and the default method, use the **--help** or **--version** switch. SiLK can support the following *COMP_METHOD* values when the required libraries are available.

none

Do not compress the output using an external library.

zlib

Use the **zlib(3)** library for compressing the output, and always compress the output regardless of the destination. Using zlib produces the smallest output files at the cost of speed.

lzo1x

Use the *lzo1x* algorithm from the LZO real time compression library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead.

snappy

Use the *snappy* library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead. *Since SiLK 3.13.0.*

best

Use lzo1x if available, otherwise use snappy if available, otherwise use zlib if available. Only compress the output when writing to a file.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwmatch** searches for the site configuration file in the locations specified in the FILES section.

--help

Print the available options and exit.

--help-relate

Print the description and aliases of each field that may be used as arguments to the **--relate** switch and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

Matching TCP Flows

rwmatch is a generalized matching tool; the most basic function provided by **rwmatch** is the ability to match both sides of a TCP connection. Given incoming and outgoing web traffic in two files *web_in.rw* and *web_out.rw*, the following sequence of commands will generate a file, *web-sessions.rw* consisting of matched sessions for every complete web session in *web_in.rw* and *web_out.rw*:

```
$ rwsort --field=1,2,3,4,stime web_in.rw  > web_in-s.rw
$ rwsort --field=2,1,4,3,stime web_out.rw > web_out-s.rw

$ rwmatch --relate=1,2 --relate=2,1 --relate=3,4 --relate=4,3      \
       web_in-s.rw  web_out-s.rw  web-sessions.rw
```

Finding Responses to a Scan

Because **rwmatch** can match fields arbitrarily, you can also match records across different protocols. Suppose there are two SiLK Flow files, *indata.rw* and *outdata.rw*, that contain the incoming and outgoing data, respectively, for a particular time period.

To trace responses to a scan attempt, we start by identifying a specific horizontal scan. In this example, we use an SMTP scan on TCP port 25. Assume that we have an IPset file, *smtp-scanners.set*, that contains the external IP addresses that scanned us port port 25. (Perhaps this file was obtained by using **rwscan(1)** and **rwscanquery(1)**.)

First, use **rwfilter(1)** to find the flow records matching these scan attempts in the incoming data file. Sort the output of **rwfilter** by source IP, source port, destination IP, destination port, and time, and store the results in *smtp-scans.rw*:

```
$ rwfilter --proto=6 --sip-set=smtp-scanners.set --dport=25        \
       --pass=-  indata.rw                                         \
  | rwsort --field=sip,sport,dip,dport,stime > smtp-scans.rw
```

We can identify hosts that responded to the scan (we consider a accepting the TCP connection as a response) by finding potential replies in the outgoing data file, sorting them, and storing the results in *scan-response.rw*. For this command on the outgoing data, note that we must swap source and destination from the values used for the incoming data:

```
$ rwfilter --proto=6 --dip-set=smtp-scanners.set --sport=25        \
       --pass=-  outdata.rw                                        \
  | rwsort --field=dip,dport,sip,sport,stime > scan-response.rw
```

We can now match the flow records to produce the file *matched-scans.rw*:

```
$ rwmatch --relate=1,2 --relate=3,4 --relate=2,1 --relate=4,3      \
       smtp-scans.rw  scan-response.rw  matched-scans.rw
```

The results file, *matched-scans.rw*, will contain all the exchanges between the scanning hosts and the responders on port 25. Examination of these flows may show evidence of buffer overflows, data exfiltration, or similar attacks.

Next, we want to identify responses to the scan that were produced by our routers, such as ICMP destination unreachable messages.

Use **rwfilter** to find the ICMP messages going to the scanning hosts, sort the flow records, and store the results in *icmp.rw*:

```
$ rwfilter --proto=1 --icmp-type=3 --pass=stdout  outdata.rw       \
  | rwsort --field=dip,stime > icmp.rw
```

Run **rwmatch** and match exclusively on the IP address.

```
$ rwmatch --relate=2,1  icmp.rw  smtp-scans.rw  result.rw
```

The resulting file, *result.rw* will consist of single packet flows (from *smtp-scans.rw*) with an ICMP response (from *icmp.rw*).

Similar queries can be used to identify other multiple-protocol phenomena, such as the results of a **traceroute**.

Displaying the Results

These examples assume *matched.rw* is an output file produced by **rwmatch**.

When using **rwcut(1)** to display the records in *matched.rw*, you may specify the next hop IP field (`nhIP`) to see the match identifier:

```
$ rwcut --num-rec=8 --fields=sip,sport,dip,dport,type,nhip matched.rw
            sIP|sPort|            dIP|dPort|   type|           nhIP|
    10.4.52.235|29631|192.168.233.171|   80|  inweb|        0.0.0.1|
192.168.233.171|   80|    10.4.52.235|29631| outweb|      255.0.0.1|
    10.9.77.117|29906| 192.168.184.65|   80|  inweb|        0.0.0.2|
 192.168.184.65|   80|    10.9.77.117|29906| outweb|      255.0.0.2|
  10.14.110.214|29989| 192.168.249.96|   80|  inweb|        0.0.0.3|
 192.168.249.96|   80|  10.14.110.214|29989| outweb|      255.0.0.3|
    10.18.66.79|29660| 192.168.254.69|   80|  inweb|        0.0.0.4|
 192.168.254.69|   80|    10.18.66.79|29660| outweb|      255.0.0.4|
```

The first record is a query from the external host 10.4.52.235 to the web server on the internal host 192.168.233.171, and the second record is the web server's response. The third and fourth records represent another query/response pair.

The **cutmatch(3)** plug-in is an alternate way to display the match parameter that **rwmatch** writes into the next hop IP field. The **cutmatch** plug-in defines a `match` field that displays the direction of the flow (-> represents a query and <- a response) and the match ID. To use the plug-in, you must explicit load it into **rwcut** by specifying the **--plugin** switch. You can then add **match** to the list of **--fields** to print:

```
$ rwcut --plugin=cutmatch.so --num-rec=8  \
       --fields=sip,sport,match,dip,dport,type matched.rw
```

```
            sIP|sPort| <->Match#|            dIP|dPort|   type|
    10.4.52.235|29631|->       1|192.168.233.171|   80|  inweb|
192.168.233.171|   80|<-       1|    10.4.52.235|29631| outweb|
    10.9.77.117|29906|->       2| 192.168.184.65|   80|  inweb|
 192.168.184.65|   80|<-       2|    10.9.77.117|29906| outweb|
  10.14.110.214|29989|->       3| 192.168.249.96|   80|  inweb|
 192.168.249.96|   80|<-       3|  10.14.110.214|29989| outweb|
    10.18.66.79|29660|->       4| 192.168.254.69|   80|  inweb|
 192.168.254.69|   80|<-       4|    10.18.66.79|29660| outweb|
```

Using the `sIP` and `dIP` fields is confusing when the file you are examining contains both incoming and outgoing flow records. To make the output from **rwmatch** more clear, use the **int-ext-fields(3)** plug-in as well. That plug-in allows you to display the external IPs in one column and the internal IPs in a another column. See its manual page for additional information.

```
$ export INCOMING_FLOWTYPES=all/in,all/inweb
$ export OUTGOING_FLOWTYPES=all/out,all/outweb
$ rwcut --plugin=cutmatch.so --plugin=int-ext-fields.so --num-rec=8 \
     --fields=ext-ip,ext-port,match,int-ip,int-port,proto matched.rw
        ext-ip|ext-p| <->Match#|         int-ip|int-p|   type|
   10.4.52.235|29631|->       1|192.168.233.171|   80|  inweb|
   10.4.52.235|29631|<-       1|192.168.233.171|   80| outweb|
   10.9.77.117|29906|->       2| 192.168.184.65|   80|  inweb|
   10.9.77.117|29906|<-       2| 192.168.184.65|   80| outweb|
  10.14.110.214|29989|->       3| 192.168.249.96|   80|  inweb|
  10.14.110.214|29989|<-       3| 192.168.249.96|   80| outweb|
    10.18.66.79|29660|->       4| 192.168.254.69|   80|  inweb|
    10.18.66.79|29660|<-       4| 192.168.254.69|   80| outweb|
```

ENVIRONMENT

SILK_IPV6_POLICY

This environment variable is used as the value for **--ipv6-policy** when that switch is not provided.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_COMPRESSION_METHOD

This environment variable is used as the value for **--compression-method** when that switch is not provided. *Since SiLK 3.13.0.*

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwmatch** may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **rwmatch** may use this environment variable. See the FILES section for details.

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

SEE ALSO

rwfilter(1), **rwsort(1)**, **rwcut(1)**, **rwfileinfo(1)**, **rwscan(1)**, **rwscanquery(1)**, **cutmatch(3)**, **int-ext-fields(3)**, **sensor.conf(5)**, **silk(7)**, **zlib(3)**

NOTES

SiLK 3.9.0 expanded the set of fields accepted by the **--relate** switch and added support for IPv6 flow records.

rwnetmask

Zero out lower bits of IP addresses in SiLK Flow records

SYNOPSIS

```
rwnetmask [--4sip-prefix-length=N] [--6sip-prefix-length=N]
      [--4dip-prefix-length=N] [--6dip-prefix-length=N]
      [--4nhip-prefix-length=N] [--6nhip-prefix-length=N]
      [--sip-prefix-length=N] [--dip-prefix-length=N]
      [--nhip-prefix-length=N] [--output-path=PATH]
      [--print-filenames] [--ipv6-policy=POLICY]
      [--note-add=TEXT] [--note-file-add=FILE]
      [--compression-method=COMP_METHOD]
      [--site-config-file=FILENAME]
      {[--xargs] | [--xargs=FILENAME] | [FILE [FILE ...]]}

rwnetmask --help

rwnetmask --version
```

DESCRIPTION

rwnetmask reads SiLK Flow records, sets the prefix of the source IP, destination IP, and/or next hop IP to the specified value(s) by masking the least significant bits of the address(es), and writes the modified SiLK Flow records to the specified output path. Modifying the IP addresses allows one to group IPs into arbitrary CIDR blocks. Multiple prefix-lengths may be specified; at least one must be specified.

When SiLK is compiled with IPv6 support, a separate mask can be specified for IPv4 and IPv6 addresses. Records are processed using the IP-version in which they are read. The **--ipv6-policy** switch can be used to force the records into a particular IP-version or to ignore records of a particular IP-version.

rwnetmask reads SiLK Flow records from the files named on the command line or from the standard input when no file names are specified and **--xargs** is not present. To read the standard input in addition to the named files, use `-` or `stdin` as a file name. If an input file name ends in `.gz`, the file is uncompressed as it is read. When the **--xargs** switch is provided, **rwnetmask** reads the names of the files to process from the named text file or from the standard input if no file name argument is provided to the switch. The input to **--xargs** must contain one file name per line.

When no output path is specified and the standard output is not connected to a terminal, **rwnetmask** writes the records to the standard output.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

One of these switches must be provided:

--4sip-prefix-length=*N*

--sip-prefix-length=*N*

For IPv4 addresses, specify the number of most significant bits of the source address to keep. The default is to not mask off any bits (i.e., N=32).

--4dip-prefix-length=*N*

--dip-prefix-length=*N*

For IPv4 addresses, specify the number of most significant bits of the destination address to keep. The default is to not mask off any bits (i.e., N=32).

--4nhip-prefix-length=*N*

--nhip-prefix-length=*N*

For IPv4 addresses, specify the number of most significant bits of the next-hop address to keep. The default is to not mask off any bits (i.e., N=32).

--6sip-prefix-length=*N*

For IPv6 addresses, specify the number of most significant bits of the source address to keep. The default is to not mask off any bits (i.e., N=128).

--6dip-prefix-length=*N*

For IPv6 addresses, specify the number of most significant bits of the destination address to keep. The default is to not mask off any bits (i.e., N=128).

--6nhip-prefix-length=*N*

For IPv6 addresses, specify the number of most significant bits of the next-hop address to keep. The default is to not mask off any bits (i.e., N=128).

These switches are optional:

--output-path=*PATH*

Write the binary SiLK Flow records to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output. If *PATH* names an existing file, **rwrwnetmask** exits with an error unless the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. If this switch is not given, the output is written to the standard output. Attempting to write the binary output to a terminal causes **rwrwnetmask** to exit with an error.

--print-filenames

Print to the standard error the names of the input files as the files are opened.

--ipv6-policy=*POLICY*

Determine how IPv4 and IPv6 flows are handled when SiLK has been compiled with IPv6 support. When the switch is not provided, the SILK_IPV6_POLICY environment variable is checked for a policy. If it is also unset or contains an invalid policy, the *POLICY* is **mix**. When SiLK has not been compiled with IPv6 support, IPv6 flows are always ignored, regardless of the value passed to this switch or in the SILK_IPV6_POLICY variable. The supported values for *POLICY* are:

ignore

Ignore any flow record marked as IPv6, regardless of the IP addresses it contains. Only records marked as IPv4 will be processed.

asv4
: Convert IPv6 flow records that contain addresses in the ::ffff:0:0/96 netblock (that is, IPv4-mapped IPv6 addresses) to IPv4 and ignore all other IPv6 flow records.

mix
: Process the input as a mixture of IPv4 and IPv6 flows.

force
: Convert IPv4 flow records to IPv6, mapping the IPv4 addresses into the ::ffff:0:0/96 netblock.

only
: Process only flow records that are marked as IPv6 and ignore IPv4 flow records in the input.

--note-add=*TEXT*
: Add the specified *TEXT* to the header of the output file as an annotation. This switch may be repeated to add multiple annotations to a file. To view the annotations, use the **rwfileinfo(1)** tool.

--note-file-add=*FILENAME*
: Open *FILENAME* and add the contents of that file to the header of the output file as an annotation. This switch may be repeated to add multiple annotations. Currently the application makes no effort to ensure that *FILENAME* contains text; be careful that you do not attempt to add a SiLK data file as an annotation.

--compression-method=*COMP_METHOD*
: Specify the compression library to use when writing output files. If this switch is not given, the value in the SILK_COMPRESSION_METHOD environment variable is used if the value names an available compression method. When no compression method is specified, output to the standard output or to named pipes is not compressed, and output to files is compressed using the default chosen when SiLK was compiled. The valid values for *COMP_METHOD* are determined by which external libraries were found when SiLK was compiled. To see the available compression methods and the default method, use the **--help** or **--version** switch. SiLK can support the following *COMP_METHOD* values when the required libraries are available.

  none
  : Do not compress the output using an external library.

  zlib
  : Use the **zlib(3)** library for compressing the output, and always compress the output regardless of the destination. Using zlib produces the smallest output files at the cost of speed.

  lzo1x
  : Use the *lzo1x* algorithm from the LZO real time compression library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead.

  snappy
  : Use the *snappy* library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead. *Since SiLK 3.13.0.*

  best
  : Use lzo1x if available, otherwise use snappy if available, otherwise use zlib if available. Only compress the output when writing to a file.

--site-config-file=*FILENAME*
: Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwnetmask** searches for the site configuration file in the locations specified in the FILES section.

--xargs

--xargs=*FILENAME*

Read the names of the input files from *FILENAME* or from the standard input if *FILENAME* is not provided. The input is expected to have one filename per line. **rwnetmask** opens each named file in turn and reads records from it as if the filenames had been listed on the command line.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following example, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

To summarize the TCP traffic from your network to each /24 on the Internet, use:

```
$ rwfilter --type=out,outweb --proto=6 --pass=stdout       \
  | rwnetmask --dip-prefix-length 24                       \
  | rwaddrcount --use-dest --sort --print-rec
IP Address| Bytes|Packets|Records|         Start Time|...
10.10.35.0|  2345|     52|      6|01/15/2003 19:30:31|
 10.23.3.0|   118|      2|      1|01/16/2003 19:38:40|
 10.23.4.0| 20858|    263|     16|01/16/2003 16:54:25|
10.31.49.0|266920|   3885|   1092|01/11/2003 02:04:11|
10.126.7.0| 36912|    260|      9|01/16/2003 17:03:28|
....
```

ENVIRONMENT

SILK_IPV6_POLICY

This environment variable is used as the value for **--ipv6-policy** when that switch is not provided.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_COMPRESSION_METHOD

This environment variable is used as the value for **--compression-method** when that switch is not provided. *Since SiLK 3.13.0.*

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwnetmask** may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **rwnetmask** may use this environment variable. See the FILES section for details.

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

SEE ALSO

rwfileinfo(1), **silk(7)**, **zlib(3)**

rwp2yaf2silk

Convert PCAP data to SiLK Flow Records with YAF

SYNOPSIS

```
rwp2yaf2silk --in=INPUT_SPEC --out=FILE [--dry-run]
    [--yaf-program=YAF] [--yaf-args='ARG1 ARG2']
    [--rwipfix2silk-program=RWIPFIX2SILK]
    [--rwipfix2silk-args='ARG1 ARG2']

rwp2yaf2silk --help

rwp2yaf2silk --man

rwp2yaf2silk --version
```

DESCRIPTION

rwp2yaf2silk is a script to convert a **pcap(3)** file, such as that produced by **tcpdump(1)**, to a single file of SiLK Flow records. The script assumes that the **yaf(1)** and **rwipfix2silk(1)** commands are available on your system.

The **--in** and **--out** switches are required. Note that the **--in** switch is processed by **yaf**, and the **--out** switch is processed by **rwipfix2silk**.

For information on reading live pcap data and using **rwflowpack(8)** to store that data in hourly files, see the *SiLK Installation Handbook*.

Normally **yaf** groups multiple packets into flow records. You can almost force **yaf** to create a flow record for every packet so that its output is similar to that of **rwptoflow(1)**: When you give **yaf** the **--idle-timeout=0** switch, **yaf** creates a flow record for every complete packet and for each packet that it is able to completely reassemble from packet fragments. Any fragmented packets that **yaf** cannot reassemble are dropped.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--in=*INPUT_SPEC*

Read the pcap records from *INPUT_SPEC*. Often *INPUT_SPEC* is the name of the pcap file to read or the string string `-` or `stdin` to read from standard input. To process multiple pcap files, create a text file that lists the names of the pcap files. Specify the text file as *INPUT_SPEC* and use `--yaf-args=--caplist` to tell **yaf** the *INPUT_SPEC* contains the names of pcap files.

--out=*FILE*

Write the SiLK Flow records to *FILE*. The string `stdout` or `-` may be used for the standard output, as long as it is not connected to a terminal.

--dry-run

Do not invoke any commands, just print the commands that would be invoked.

--yaf-program=*YAF*

Use *YAF* as the location of the **yaf** program. When not specified, **rwp2yaf2silk** assumes there is a program **yaf** on your $PATH.

--yaf-args=*ARGS*

Pass the additional *ARGS* to the **yaf** program.

--rwipfix2silk-program=*RWIPFIX2SILK*

Use *RWIPFIX2SILK* as the location of the **rwipfix2silk** program. When not specified, **rwp2yaf2silk** assumes there is a program **rwipfix2silk** on your $PATH.

--rwipfix2silk-args=*ARGS*

Pass the additional *ARGS* to the **rwipfix2silk** program.

--help

Display a brief usage message and exit.

--man

Display full documentation for **rwp2yaf2silk** and exit.

--version

Print the version number and exit the application.

SEE ALSO

yaf(1), **rwipfix2silk(1)**, **rwflowpack(8)**, **rwptoflow(1)**, **silk(7)**, **tcpdump(1)**, **pcap(3)**, *SiLK Installation Handbook*

rwpcut

Outputs a tcpdump dump file as ASCII

SYNOPSIS

```
rwpcut [--columnar]
       [--delimiter=DELIMITER]
       [--epoch-time]
       [--fields=PRINT_FIELDS]
       [--integer-ips]
       [--zero-pad-ips]
       FILE...
```

DESCRIPTION

rwpcut outputs tcpdump files in an easy to parse way. It supports a user-defined list of fields to output and a user-defined delimiter between columns.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option.

OUTPUT SWITCHES

--columnar

Pad each field with whitespace so that it always takes up the same number of columns. The two payload printing fields, payhex and payascii, never pad with whitespace.

--delimiter=*DELIMITER*

DELIMITER is used as the delimiter between columns instead of the default '|'.

--epoch-time

Display the timestamp as epoch time seconds instead of a formatted timestamp.

--fields=*PRINT_FIELDS*

PRINT_FIELDS is a comma-separated list of fields to include in the output. The available fields are:

timestamp - packet timestamp sip - source IP address. dip - destination IP address sport - source port dport - destination port proto - IP protocol payhex - Payload printed as a hex stream payascii - Payload printed as an ascii stream. Non-printing characters are represented with periods.

--integer-ips

Display IP addresses as integers instead of in dotted quad notation.

--zero-pad-ips

Pad dotted quad notation IP addresses so that each quad occupies three columns.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line.

```
$ rwpcut --fields=sip,dip,sport,dport,proto --columnar data.dmp

                sip|            dip|sport|dport|proto|
   220.245.221.126|  192.168.1.100|21776| 6882|    6|
   220.245.221.126|  192.168.1.100|21776| 6882|    6|

$ rwpcut --fields=timestamp,payhex data.dmp
```

(Carriage returns mid-payload added for legibility)

```
   timestamp|payhex|
   2005-04-20 04:28:59.091470|4500003cd85840003206f3e2dcf5dd7
   ec0a8016455101ae2811b6bce00000000a002ffff59990000020405ac0
   10303000101080a524dc5cc00000000|
   2005-04-20 04:29:02.057390|4500003cd88c40003206f3aedcf5dd7
   ec0a8016455101ae2811b6bce00000000a002ffff59930000020405ac0
   10303000101080a524dc5d200000000|
```

SEE ALSO

rwptoflow(1), **silk(7)**

BUGS

Note that payhex and payascii do not whitespace pad themselves if **--columnar** is used.

The payascii field does not escape the delimiter character in any way, so care should be taken when parsing it.

rwpdedupe

Eliminate duplicate packets collected by several sensors

SYNOPSIS

```
rwpdedupe { --first-duplicate | --random-duplicate[=SCALAR] }
      [--threshold=MILLISECONDS] FILE... > OUTPUT-FILE

rwpdedupe --help

rwpdedupe --version
```

DESCRIPTION

Detects and eliminates duplicate records from **tcpdump(1)** capture files. Duplicate records are defined as having timestamps within a user-configurable time of each other. In addition, their Ethernet (OSI layer 2) headers must match. If they are not IP packets, then their entire Ethernet payload must match. If they are IP packets, then their source and destination addresses, protocol, and IP payload must match.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--threshold=*MILLISECONDS*

Set the maximum number of milliseconds which may elapse between two packets and still have those packets be detected as duplicates. Default 0 (exact timestamp match). Must be a value between 0 and 1,000,000 milliseconds.

One and only one of the following switches is required:

--first-duplicate

When selecting between multiple duplicate packets, always choose the packet with the earliest timestamp. Not compatible with **--random-duplicate**.

--random-duplicate

--random-duplicate=*SCALAR*

Select a random packet from the list of duplicate packets. SCALAR is a random number seed, so that multiple runs can produce identical results.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following example, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

Given **tcpdump** files *data1.tcp* and *data2.tcp*, detect and eliminate duplicate packets which occur within one second of each other (when choosing which timestamp to output, pick one randomly.) Store the result file in *out.tcp*.

```
$ rwpdedupe --threshold=1000 --random-duplicate \
        data1.tcp data2.tcp > out.tcp
```

SEE ALSO

silk(7), **mergecap(1)**, **tcpdump(1)**, **pcap(3)**

NOTES

mergecap(1) can be used to merge two **tcpdump** capture files without eliminating duplicate packets.

rwpdu2silk

Convert NetFlow v5 records to SiLK Flow records

SYNOPSIS

```
rwpdu2silk [--silk-output=PATH] [--print-statistics]
      [--log-destination={stdout | stderr | none | PATH}]
      [--log-flags={none | { {all | bad | default | missing
                            | record-timestamps} ...} } ]
      [--note-add=TEXT] [--note-file-add=FILE]
      [--compression-method=COMP_METHOD]
      {--xargs | --xargs=FILENAME | PDUFILE [PDUFILE...]}

rwpdu2silk --help

rwpdu2silk --version
```

DESCRIPTION

rwpdu2silk reads NetFlow v5 PDU (Protocol Data Units) records from one or more files, converts the records to the SiLK Flow format, and writes the SiLK records to the path specified by **--silk-output** or to the standard output when **--silk-output** is not provided. Note that **rwpdu2silk** cannot read from the standard input.

rwpdu2silk expects its input files to be a the format created by Cisco's NetFlow Collector: The file's size must be an integer multiple of 1464, where each 1464 octet chunk contains a 24 octet NetFlow v5 header and space for thirty 48 octet NetFlow v5 records. The number of valid records per chunk is specified in the header.

rwpdu2silk reads NetFlow v5 records from the files named on the command line when **--xargs** is not present. If an input file name ends in `.gz`, the file is uncompressed as it is read. When the **--xargs** switch is provided, **rwpdu2silk** reads the names of the files to process from the named text file or from the standard input if no file name argument is provided to the switch. The input to **--xargs** must contain one file name per line.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--silk-output=*PATH*

Write the binary SiLK Flow records to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output. If *PATH* names an existing file, **rwpdu2silk** exits with an error unless the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. If this switch is not given, the output is written to the standard output. Attempting to write the binary output to a terminal causes **rwpdu2silk** to exit with an error.

--print-statistics

Print, to the standard error, the number of records that were written to the SiLK output file. See also **--log-destination**.

--log-destination={none | stdout | stderr | *PATH*}

Write more detailed information to the specified destination. The default destination is **none** which suppresses messages. Use **stdout** or **stderr** to send messages to the standard output or standard error, respectively. Any other value is treated as a file name in which to write the messages. When an existing file is specified, **rwpdu2silk** appends any messages to the file. Information that is written includes the following:

- For each input stream, the number of PDU records read, number of SiLK records generated, number of missing records (based on the NetFlow v5 sequence number), and number of invalid records.
- Messages about each NetFlow v5 packet that was rejected due a bad version number or having a record count of 0 or more than 30.
- Additional messages enabled by the **--log-flags** switch.

--log-flags=*FLAGS*

Write additional messages regarding the NetFlow v5 data to the **--log-destination**, where *FLAGS* is a comma-separated list of names specifying the type messages to write. When this switch is not specified, the default value for *FLAGS* is `none`. This switch takes the same values as the **log-flags** setting in the **sensor.conf(5)** file. This manual page documents the values that are relevant for NetFlow v5 data. *Since SiLK 3.10.0.*

all

Log everything.

bad

Write messages about an individual NetFlow v5 record where the packet or octet count is zero, the packet count is larger than the octet count, or the duration of the flow is larger than 45 days.

default

Enable the default set of log-flags used by *sensor.conf*: **bad**, **missing**. Despite the name, this is not the default setting for this switch; **none** is.

missing

Examine the sequence numbers of NetFlow v5 packets and write messages about missing and out-of-sequence packets.

none

Log nothing. It is an error to combine this log-flag name with any other. This is the default setting for **--log-flags**.

record-timestamps

Log the timestamps that appear on each record. This produces a lot of output, and it is primarily used for debugging.

--note-add=*TEXT*

Add the specified *TEXT* to the header of the output file as an annotation. This switch may be repeated to add multiple annotations to a file. To view the annotations, use the **rwfileinfo(1)** tool.

--note-file-add=*FILENAME*

Open *FILENAME* and add the contents of that file to the header of the output file as an annotation. This switch may be repeated to add multiple annotations. Currently the application makes no effort to ensure that *FILENAME* contains text; be careful that you do not attempt to add a SiLK data file as an annotation.

--compression-method=*COMP_METHOD*

Specify the compression library to use when writing output files. If this switch is not given, the value in the SILK_COMPRESSION_METHOD environment variable is used if the value names an available compression method. When no compression method is specified, output to the standard output or to named pipes is not compressed, and output to files is compressed using the default chosen when SiLK was compiled. The valid values for *COMP_METHOD* are determined by which external libraries were found when SiLK was compiled. To see the available compression methods and the default method, use the **--help** or **--version** switch. SiLK can support the following *COMP_METHOD* values when the required libraries are available.

none

Do not compress the output using an external library.

zlib

Use the **zlib(3)** library for compressing the output, and always compress the output regardless of the destination. Using zlib produces the smallest output files at the cost of speed.

lzo1x

Use the *lzo1x* algorithm from the LZO real time compression library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead.

snappy

Use the *snappy* library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead. *Since SiLK 3.13.0.*

best

Use lzo1x if available, otherwise use snappy if available, otherwise use zlib if available. Only compress the output when writing to a file.

--xargs

--xargs=*FILENAME*

Read the names of the input files from *FILENAME* or from the standard input if *FILENAME* is not provided. The input is expected to have one filename per line. **rwpdu2silk** opens each named file in turn and reads records from it as if the filenames had been listed on the command line.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

ENVIRONMENT

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_COMPRESSION_METHOD

This environment variable is used as the value for **--compression-method** when that switch is not provided. *Since SiLK 3.13.0.*

SEE ALSO

rwfileinfo(1), **rwflowpack(8)**, **sensor.conf(5)**, **silk(7)**, **zlib(3)**

BUGS

rwpdu2silk cannot read from the standard input.

rwpmapbuild

Create a binary prefix map from a text file

SYNOPSIS

```
rwpmapbuild [--input-path=PATH] [--output-path=PATH]
      [--mode={ipv4|ipv6|proto-port}] [--dry-run] [--ignore-errors]
      [--note-add=TEXT] [--note-file-add=FILENAME]
      [--invocation-strip]

rwpmapbuild --help

rwpmapbuild --version
```

DESCRIPTION

rwpmapbuild reads a white-space-delimited stream of text and writes a binary output stream representing a prefix map. The syntax of this input is described in the INPUT FILE FORMAT section below.

The textual input is read from the file specified by **--input-path** or from the standard input when the switch is not provided. The binary output is written to the location named by **--output-path** or to the standard output when the switch is not provided and the standard output is not connected to a terminal.

A prefix map file is a binary file that maps a value (specifically either an IP addresses or a protocol-port pair) to a string label.

Once you have created a prefix map file, you may use the file in **rwfilter(1)**, **rwstats(1)**, **rwuniq(1)**, **rwgroup(1)**, **rwsort(1)**, or **rwcut(1)** to partition, count, sort and display SiLK flow records based on the string labels defined in the prefix map. See the **pmapfilter(3)** manual page for details. To view the contents of a prefix map file, use **rwpmapcat(1)**. To query the contents of a prefix map, use **rwpmaplookup(1)**.

The remainder of this section provides example files that could be used as input to **rwpmapbuild**, and a note on the proper ordering of the input. For details on the syntax of the input, see the INPUT FILE FORMAT section that follows the description of the command line OPTIONS.

Sample IPv4 input file

The following is a sample input file for **rwpmapbuild** that describes the registry of special-purpose IPv4 addresses. Any IP address that is not a special-purpose address get the label `Normal`.

```
# Prefix map sample input file for special purpose IPv4 addresses
map-name   ipv4-special
mode       ipv4
default    Normal

#    Each line has an either a CIDR block or a pair of IP
#    addresses and then a label for that range
0.0.0.0/8          This host on this network [RFC1122 section 3.2.1.3]
10.0.0.0/8         Private-Use [RFC1918]
```

```
100.64.0.0/10      Shared Address Space [RFC6598]
127.0.0.0/8        Loopback [RFC1122 section 3.2.1.3]
169.254.0.0/16     Link Local [RFC3927]
172.16.0.0/12      Private-Use [RFC1918]
192.0.0.0/24       IETF Protocol Assignments [RFC6890 section 2.1]
192.0.0.0/29       IPv4 Service Continuity Prefix [RFC7335]
#    A range of a single IP address requires a "/32" suffix or
#    that the IP address be repeated
192.0.0.8/32       IPv4 dummy address [RFC7600]
192.0.0.9/32       Port Control Protocol Anycast [RFC7723]
192.0.0.10/32      Traversal Using Relays around NAT Anycast [draft]
#    A range may be specified as two IP addresses
192.0.0.170  192.0.0.171   NAT64/DNS64 Discovery [RFC7050 section 2.2]
192.0.2.0/24       Documentation (TEST-NET-1) [RFC5737]
192.31.196.0/24    AS112-v4 [RFC7535]
192.52.193.0/24    AMT [RFC7450]
192.88.99.0/24     Deprecated (6to4 Relay Anycast) [RFC7526]
192.168.0.0/16     Private-Use [RFC1918]
192.175.48.0/24    Direct Delegation AS112 Service [RFC7534]
198.18.0.0/15      Benchmarking [RFC2544]
198.51.100.0/24    Documentation (TEST-NET-2) [RFC5737]
203.0.113.0/24     Documentation (TEST-NET-3) [RFC5737]
240.0.0.0/4        Reserved [RFC1112 section 4]
255.255.255.255/32 Limited Broadcast [RFC919 section 7]
```

Sample IPv6 input file

The following input file for **rwpmapbuild** describes IPv6 address space. The file demonstrates the use of the **label** statement. It uses a hyphen ("-") as the label for any undefined ranges.

```
# Prefix map sample input file for IPv6 address space
map-name   iana-ipv6
mode       ipv6

label      0   RFC3849
label      1   RFC3879
label      2   RFC4048
label      3   RFC4193
label      4   RFC4291
label      5   RFC4291 Loopback Address
label      6   RFC4291 Unspecified Address
label      7   RFC4291 IPv4-mapped Address
label      8   RFC5180
label      9   RFC6666
label     10   RFC7723
label     11   -

default        -

0000::/8       4
```

```
::1/128        5
::/128         6
::ffff:0:0/96  7
0100::/8       4
0100::/64      9           # RFC6666
0200::/7       2           # RFC4048
0400::/6       4
0800::/5       4
1000::/4       4
2000::/3       4
2001:1::1/128 10           # RFC7723
2001:2::/48    8           # Benchmarking
2001:db8::/32  0           # Documentation
4000::/3       4
6000::/3       4
8000::/3       4
a000::/3       4
c000::/3       4
e000::/4       4           # You may use the label number or the
f000::/5       RFC4291     # exact label name, but any other text
f800::/6       4           # causes rwpmapbuild to issue an error
fc00::/7       RFC4193
fe00::/9       4
fe80::/10      4
fec0::/10      RFC3879
ff00::/8       4
```

Sample protocol-port input file

This is a small sample of a file that could be used to label IP protocols, specific ports within the TCP and UDP protocols, and ICMP type and code values. When ranges overlap or one range is a specialization of another, the wider or more general range should be listed first, followed by the narrower or more specific ranges.

```
map-name   protocol-port-example
mode       proto-port

#    The range is either a single protocol or a protocol and
#    a port separated by a slash.
1      1           ICMP
#    Specify the wider categories first, then specialize
6      6           TCP
6/0    6/1024      TCP/Generic reserved
#    A range of a single port requires both the starting
#    value and the ending value
6/21   6/21        TCP/FTP
6/22   6/22        TCP/SSH
6/25   6/25        TCP/SMTP
6/80   6/80        TCP/HTTP
6/443  6/443       TCP/HTTPS
6/6000 6/6063      TCP/X11
```

```
17     17          UDP
17/0   17/1024     UDP/Generic reserved
17/53  17/53       UDP/DNS
17/67  17/68       UDP/DHCP
50     50          ESP
58     58          ICMPv6
#    For ICMP Type/Code, convert the type and code to a port
#    value using this expression:  type * 256 + code
1/0    1/255       ICMP/Echo Reply
1/768  1/1023      ICMP/Destination Unreachable
1/1024 1/1279      ICMP/Source Quench
1/768  1/768       ICMP/Net Unreachable
1/769  1/769       ICMP/Host Unreachable
1/770  1/770       ICMP/Protocol Unreachable
1/771  1/771       ICMP/Port Unreachable
```

Complete ICMPv4 Prefix Map

An ideal candidate for port-based prefix maps is for decoding ICMP types and codes. Although most SiLK commands support a form of ICMP type and code options, these are all based on the actual number values. However, a prefix map may be useful to decode the noun-name of the ICMP types and codes. The following prefix map can be used for that purpose. (Note that ICMP type and code is always in the destination port field, regardless of the traffic direction.)

```
# Identify this as a protocol-port prefix map, rather than an IP-range map
mode proto-port


# Set a default value for all records
0 255 Other


# Set the default value for all ICMP records
1 1 ICMP/Undefined


# ICMP specific entries
1/0 1/255 ICMP/Echo Reply
1/768 1/768 ICMP/Destination Unreachable/Net Unreachable
1/769 1/769 ICMP/Destination Unreachable/Host Unreachable
1/770 1/770 ICMP/Destination Unreachable/Protocol Unreachable
1/771 1/771 ICMP/Destination Unreachable/Port Unreachable
1/772 1/772 ICMP/Destination Unreachable/Fragmentation Needed and Don't Fragment was Set
1/773 1/773 ICMP/Destination Unreachable/Source Route Failed
1/774 1/774 ICMP/Destination Unreachable/Destination Network Unknown
1/775 1/775 ICMP/Destination Unreachable/Destination Host Unknown
1/776 1/776 ICMP/Destination Unreachable/Source Host Isolated
1/777 1/777 ICMP/Destination Unreachable/Communication with Destination Network is Administratively Pr
1/778 1/778 ICMP/Administratively Prohibited/Communication with Destination Host is Administratively P
1/779 1/779 ICMP/Administratively Prohibited/Destination Network Unreachable for Type of Service
1/780 1/780 ICMP/Administratively Prohibited/Destination Host Unreachable for Type of Service
1/781 1/781 ICMP/Administratively Prohibited/Communication Administratively Prohibited
1/782 1/782 ICMP/Administratively Prohibited/Host Precedence Violation
```

```
1/783 1/783 ICMP/Administratively Prohibited/Precedence cutoff in effect
1/1024 1/1279 ICMP/Source Quench
1/1280 1/1280 ICMP/Redirect/Redirect Datagram for the Network (or subnet)
1/1281 1/1281 ICMP/Redirect/Redirect Datagram for the Host
1/1282 1/1282 ICMP/Redirect/Redirect Datagram for the Type of Service and Network
1/1283 1/1283 ICMP/Redirect/Redirect Datagram for the Type of Service and Host
1/1536 1/1536 ICMP/Alternate Host Address/Alternate Address for Host
1/2048 1/2303 ICMP/Echo
1/2304 1/2304 ICMP/Router Advertisement/Normal router advertisement
1/2320 1/2320 ICMP/Router Advertisement/Does not route common traffic
1/2560 1/2815 ICMP/Router Selection
1/2816 1/2816 ICMP/Time Exceeded/Time to Live exceeded in Transit
1/2817 1/2817 ICMP/Time Exceeded/Fragment Reassembly Time Exceeded
1/3072 1/3072 ICMP/Parameter Problem/Pointer indicates the error
1/3073 1/3073 ICMP/Parameter Problem/Missing a Required Option
1/3074 1/3074 ICMP/Parameter Problem/Bad Length
1/3328 1/3583 ICMP/Timestamp
1/3584 1/3839 ICMP/Timestamp Reply
1/3840 1/4095 ICMP/Information Request
1/4096 1/4351 ICMP/Information Reply
1/4352 1/4607 ICMP/Address Mask Request
1/4608 1/4863 ICMP/Address Mask Reply
1/7680 1/7935 ICMP/Traceroute
1/7936 1/8191 ICMP/Datagram Conversion Error
1/8192 1/8447 ICMP/Mobile Host Redirect
1/8448 1/8703 ICMP/IPv6 Where-Are-You
1/8704 1/8959 ICMP/IPv6 I-Am-Here
1/8960 1/9215 ICMP/Mobile Registration Request
1/9216 1/9471 ICMP/Mobile Registration Reply
1/9984 1/10239 ICMP/SKIP
1/10240 1/10240 ICMP/Photuris/Bad SPI
1/10241 1/10241 ICMP/Photuris/Authentication Failed
1/10242 1/10242 ICMP/Photuris/Decompression Failed
1/10243 1/10243 ICMP/Photuris/Decryption Failed
1/10244 1/10244 ICMP/Photuris/Need Authentication
1/10245 1/10245 ICMP/Photuris/Need Authorization


# A few other well-known protocols
6 6 TCP
17 17 UDP
50 50 ESP
51 51 AH
```

Proper Ordering of rwpmapbuild Input

When creating the textual input for **rwmpabuild**, be sure to put the most general attributes first in the list.

For example, suppose we administer the address block 12.0.0.0/8, and would like to report on address ranges delegated within the organization, A prefix map can be used as follows to show utilization for each address block, as well as unallocated (and presumably unauthorized) usage.

Display the contents of input file:

```
$ cat network.pmap.txt
12.0.0.0/8 Assigned, Unallocated
12.1.0.0/16 RESERVED
12.38.0.0/16 Client Network 1
12.127.0.0/16 Data Center (Primary)
12.130.0.0/16 Client Network 2
12.154.0.0/16 Client Network 3
12.186.0.0/16 Data Center (Secondary)
12.210.0.0/16 RESERVED
```

Create the binary prefix map:

```
$ rwpmapbuild --input=network.pmap.txt --output=network.pmap
```

Use **rwfilter(1)** to select IPs in the 12.0.0.0/8 netblock, and use **rwuniq(1)** to bin the results according to the prefix map:

```
$ rwfilter --start=2007/07/30:00 --saddr=12.x.x.x --pass=stdout  \
  | rwuniq --pmap-file=network.pmap --field=sval --value=bytes

                  sval|               Bytes|
              RESERVED|               39749|
  Data Center (Primary)|               87621|
  Assigned, Unallocated|             4296212|
       Client Network 2|              545848|
Data Center (Secondary)|               18228|
       Client Network 1|              112404|
       Client Network 3|               68820|
```

Suppose the input file had placed the most general entry at the bottom, like so:

```
$ cat network.pmap.txt
12.1.0.0/16 RESERVED
12.38.0.0/16 Client Network 1
12.127.0.0/16 Data Center (Primary)
12.130.0.0/16 Client Network 2
12.154.0.0/16 Client Network 3
12.186.0.0/16 Data Center (Secondary)
12.210.0.0/16 RESERVED
12.0.0.0/8 Assigned, Unallocated
```

The general entry would overwrite the other entries:

```
$ rwpmapbuild --input=network.pmap.txt --output=network.pmap
$ rwfilter --start=2007/07/30:00 --saddr=12.x.x.x --pass=stdout \
  | rwuniq --pmap-file=network.pmap --fields=sval --bytes
                  sval|               Bytes|
  Assigned, Unallocated|             5168882|
```

This is easy to see by using **rwpmapcat(1)** to print the contents of the prefix map:

```
$ rwpmapcat --map-file=network.pmap
         ipBlock|                  label|
       0.0.0.0/5|                UNKNOWN|
       8.0.0.0/6|                UNKNOWN|
      12.0.0.0/8|  Assigned, Unallocated|
      13.0.0.0/8|                UNKNOWN|
      14.0.0.0/7|                UNKNOWN|
      16.0.0.0/4|                UNKNOWN|
      32.0.0.0/3|                UNKNOWN|
      64.0.0.0/2|                UNKNOWN|
     128.0.0.0/1|                UNKNOWN|
```

(**rwpmapcat** lists all possible addresses from 0.0.0.0 to 255.255.255.255 and their labels. The default label is `UNKNOWN` unless the default is set to something else.)

The best way to make sure your entries are properly ordered is to explicitly order them before compiling the prefix map. When the data uses the CIDR-block format, the UNIX **sort(1)** command often produces the proper output.

```
$ cat network.pmap.txt
12.1.0.0/16 RESERVED
12.38.0.0/16 Client Network 1
12.127.0.0/16 Data Center (Primary)
12.130.0.0/16 Client Network 2
12.154.0.0/16 Client Network 3
12.186.0.0/16 Data Center (Secondary)
12.210.0.0/16 RESERVED
12.0.0.0/8 Assigned, Unallocated
```

Split the input at the `/` and sort the input numerically by the bitmask size. (A small bitmask represents a large netblock.)

```
$ sort -n -k 2 -t "/" network.pmap.txt
12.0.0.0/8 Assigned, Unallocated
12.1.0.0/16 RESERVED
12.127.0.0/16 Data Center (Primary)
12.130.0.0/16 Client Network 2
12.154.0.0/16 Client Network 3
12.186.0.0/16 Data Center (Secondary)
12.210.0.0/16 RESERVED
12.38.0.0/16 Client Network 1
```

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--input-path=*PATH*

Read the textual input from *PATH*. You may use `stdin` or `-` to represent the standard input. When this switch is not provided, the input is read from the standard input unless the standard input is a terminal. **rwpmapbuild** will read textual input from the terminal if the standard input is explicitly specified as the input. The input file format is described below. (Added in SiLK 3.17.0 as a replacement for **--input-file**.)

--output-path=*PATH*

Write the binary prefix map to *PATH*. You may use `stdout` or `-` to represent the standard output. When this switch is not provided, the prefix map is written to the standard output unless the standard output is connected to a terminal. (Added in SiLK 3.17.0 as a replacement for **--output-file**.)

--mode={ipv4|ipv6|proto-port}

Specify the type of the input, as if a **mode** statement appeared in the input stream. The value specified by this switch must not conflict with an explicit **mode** statement appearing in the input.

--dry-run

Do not write the output file. Simply check the syntax of the input file.

--ignore-errors

Write the output file regardless of any errors encountered while parsing the input file.

--note-add=*TEXT*

Add the specified *TEXT* to the header of the output file as an annotation. This switch may be repeated to add multiple annotations to a file. To view the annotations, use the **rwfileinfo(1)** tool.

--note-file-add=*FILENAME*

Open *FILENAME* and add the contents of that file to the header of the output file as an annotation. This switch may be repeated to add multiple annotations. Currently the application makes no effort to ensure that *FILENAME* contains text; be careful that you do not attempt to add a SiLK data file as an annotation.

--invocation-strip

Do not record the command used to create the prefix map in the output. When this switch is not given, the invocation is written to the file's header, and the invocation may be viewed with **rwfileinfo(1)**. *Since SiLK 3.12.0.*

--input-file=*PATH*

Read the textual input from *PATH*. An alias for **--input-path**. Deprecaated as of SiLK 3.17.0.

--output-file=*PATH*

Write the binary prefix map to *PATH*. An alias for **--output-path**. Deprecaated as of SiLK 3.17.0.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

INPUT FILE FORMAT

This section describes the format of the textual input file for **rwpmapbuild**. Three example files are shown above in the DESCRIPTION section.

Blank lines or lines containing only whitespace in the input file are ignored.

The file may contain comments, and these are ignored. A comment begins with the first `#` character on a line and extends to the end of the line. Note that `#` appearing in a textual label is treated as the beginning of a comment.

Each non-blank line in the input file that is not a comment is considered a statement. A statement must be completed on a single line, and only one statement may appear on a line.

The delimiter in the input file is whitespace---specifically one or more space and/or tab characters.

rwpmapbuild supports five types of statements. Four of those statements begin with a specific keyword: one of **mode**, **map-name**, **label**, and **default**. Any line that does not begin with with a keyword is expected to contain a range definition, which maps a range to a label. The format of the range definition depends on the **mode**.

The four statement types that begin with a keyword are all optional. They are:

mode { ipv4 | ipv6 | proto-port | ip }

Specify what types of ranges are defined in the file. The **mode** statement must appear before any ranges are specified. The mode may also be set using the **--mode** command line switch. When both the **mode** statement and the **--mode** switch are given, their values must match. When neither the **mode** statement nor the **--mode** switch is provided, **rwpmapbuild** processes the input in IPv4 address mode. The **ip** mode is deprecated; it is an alias for **ipv4**. The **mode** statement may only appear one time.

map-name ***simple-string***

Create a name for the data in this prefix map file that other SiLK tools may use to refer to this prefix map file. When the prefix map file is used by **rwfilter(1)**, the *simple-string* is used to generate the filtering switch names. When the prefix map file is used by **rwcut(1)**, **rwgroup(1)**, **rwsort(1)**, **rwstats(1)**, or **rwuniq(1)**, the *simple-string* is used to generate the field names. See **pmapfilter(3)** for details. The *simple-string* may not contain whitespace, a comma, or a colon. The **map-name** statement may only appear one time.

label ***num label-text***

Associate the numeric identifier *num* with the given label text *label-text*. By specifying a **label** statement, the identifier *num* is expanded to *label-text* when the range definitions are being defined.

Either all labels used in the file must appear in **label** statements, or no **label** statements may appear in which case **rwpmapbuild** creates labels as it parses the range definitions. All **label** statements must appear before the **default** statement and before the range definitions.

label-text is a textual string that begins at the first non-whitespace character and extends to the final non-whitespace character on that line that does not appear in a comment. The *label-text* may include embedded whitespace and non-alphanumeric characters. While a comma (`,`) is legal in the *label-text*, using a comma prevents the label from being used by the **--pmap-src** and **--pmap-dest** switches in **rwfilter(1)**.

If no **label** statements appear in the input, any text containing at least one non-whitespace character may be used as the label in the **default** statement and the range definitions.

It is an error if *num* or *label-text* appear in any other **label** statement. The minimum value for *num* is 0 and the maximum value is 2147483647. Note that **rwpmapbuild** creates labels for **all** numeric

identifiers between 0 and the maximum identifier used in the input file, and using an unnecessarily large value creates many empty entries.

default ***label-value***

Use the label *label-value* for any ranges not explicitly mentioned in this input file. The *label-value* text is one of

1. when **label** statements are used, a numerical label identifier that was specified in one of the statements
2. when **label** statements are used, a string that is an *exact* match of the *label-text* that was specified in one of those statements
3. when **label** statements are *not* used, a string that begins at the first non-whitespace character and extends to the final non-whitespace character on that line that does not appear in a comment

The **default** statement must appear before the range definitions are specified. If the **default** statement does not appear in the input, the label `UNKNOWN` is automatically defined and used as the default.

As mentioned above, any line that does not begin with one of the above keywords must contain a range definition, and the format of the line depends on the type of data in the input file--that is, the **mode** of the input.

Regardless of the input mode, the final item in each range definition is the label to assign to that range. The label has the same form as that described for the **default** statement above, and the label is denoted by *label-value* in the following.

Address Mode

When the **mode** is either **ipv4** or **ipv6**, **rwpmapbuild** parses the file in address mode. In address mode, each range definition contains an IP range and a *label-value*.

A range is either a CIDR block or a pair of IP addresses that specify the starting IP and ending IP of the range. To provide a label for a single IP address, you must either use the same IP address as the starting and ending values of the range, append `/32` to a single IPv4 address, or append `/128` to a single IPv6 address. When the **mode** is **ipv4**, an IPv6 address in the input file raises an error. **rwpmapbuild** also accepts integer representations of IP addresses when in **ipv4** mode,

When ranges overlap or one range is a specialization of another, the wider or more general range should be listed first, followed by the narrower or more specific ranges.

cidr-block label-value

Associate the given label identifier or label text with this CIDR block. The CIDR block is composed of an IP address in canonical notation (e.g, dotted-decimal for IPv4), a slash `/`, and the number of significant bits.

low-ip high-ip label-value

Associate the given label identifier or label text with this IP range, where *low-ip* and *high-ip* are in canonical notation.

low-int high-int label-value

Treat *low-int* and *high-int* as 32-bit values, convert the values to IPv4 addresses, and associate the given label identifier or label text with the IPv4 range.

Protocol/Port Mode

When the **mode** is **proto-port**, **rwpmapbuild** parses the file in protocol/port mode. In protocol/port mode, each range definition contains a starting value, an ending value, and the *label-value*.

The starting and ending values may both be integers between 0 and 255, inclusive. The numbers are treated as protocol values, where 6 is TCP, 17 is UDP, and 1 is ICMP.

The starting and ending values may also both be a number, a slash (`/`), and a number. The first number is treated as the protocol and the second number as a port number (or service) in that protocol. For example, 6/80 is considered the http service of TCP.

When ranges overlap or one range is a specialization of another, the wider or more general range should be listed first, followed by the narrower or more specific ranges. That is, specify the generic protocol first, then port numbers within that protocol.

proto/port proto/port label-value

Associate the given label identifier or label text with all protocols and port numbers between these two values inclusive. Note that while port is not meaningful for all protocols (specifically, it is meaningful for TCP and UDP and may contain type/code information for ICMP), **rwpmapbuild** accepts port numbers for any protocol.

proto proto label-value

Associate the given label identifier or label text for all protocols between these two values inclusive.

EXAMPLE

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

Reading from and writing to a file:

```
$ rwpmapbuild --input iana-ipv6.txt --output iana-ipv6.pmap
```

Reading from the standard input and writing to the standard output:

```
$ cat ipv4-special.txt              \
  | rwpmapbuild > ipv4-special.pmap
```

For example input files, see the DESCRIPTION section above.

ENVIRONMENT

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SEE ALSO

pmapfilter(3), **rwfilter(1)**, **rwfileinfo(1)**, **rwpmapcat(1)**, **rwpmaplookup(1)**, **rwcut(1)**, **rwgroup(1)**, **rwsort(1)**, **rwstats(1)**, **rwuniq(1)**, **silk(7)**

rwpmapcat

Print each range and label present in a prefix map file

SYNOPSIS

```
rwpmapcat [--output-types={mapname | type | ranges | labels}]
      [--ignore-label=LABEL] [--ip-label-to-ignore=IP_ADDRESS]
      [--left-justify-labels] [--no-cidr-blocks]
      [--ip-format=FORMAT] [--integer-ips] [--zero-pad-ips]
      [--no-titles] [--no-columns] [--column-separator=C]
      [--no-final-delimiter] [{--delimited | --delimited=C}]
      [--output-path=PATH] [--pager=PAGER_PROG]
      [ { --map-file=PMAP_FILE | PMAP_FILE
          | --address-types | --address-types=MAP_FILE
          | --country-codes | --country-codes=MAP_FILE } ]


rwpmapcat --help


rwpmapcat --version
```

DESCRIPTION

rwpmapcat reads a prefix map file created by **rwpmapbuild(1)** or **rwgeoip2ccmap(1)** and prints its contents.

By default, **rwpmapcat** prints the range/label pairs that exist in the prefix map. Use the **--output-types** switch to print additional information or information other than the range/label pairs.

When printing the range/label pairs of a prefix map file that contain IP address data, **rwpmapcat** defaults to printing the range as an address block in CIDR notation and the label associated with that block. To print the ranges as a starting address and ending address, specify the **--no-cidr-blocks** switch.

If the prefix map file contains protocol/port pairs, **rwpmapcat** prints three fields: the starting protocol and port separated by a slash (`/`), the ending protocol and port, and the label.

The printing of ranges having a specific label may be suppressed with the **--ignore-label** switch. To have **rwpmapcat** to look up a label based on an IP address and then ignore all entries with that label, pass the IP address to the **--ip-label-to-ignore** switch.

To print the contents of an arbitrary prefix map file, one may pipe the file to **rwpmapcat**'s standard input, name the file as the argument to the **--map-file** switch, or name the file on the command line.

To print the contents of the default country codes mapping file (see **ccfilter(3)**), specify the **--country-codes** switch with no argument. To print the contents of a specific country codes mapping file, name that file as the argument to the **--country-codes** switch.

For printing the address types mapping file (see **addrtype(3)**), use the **--address-types** switch which works similarly to the **--country-codes** switch.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

Many of options are ignored unless **rwpmapcat** is printing the range/label pairs present in the prefix map file.

--map-file=*PMAP_FILE*

Specify the path of the prefix map file to print. If this switch is omitted and neither **--country-codes** nor **--address-types** is specified, the name of the file to be read is taken as the first non-switch command-line argument. If no argument is given, **rwpmapcat** attempts to read the map from the standard input.

--address-types

Print the contents of the address types mapping file (**addrtype(3)**) specified by the SILK_ADDRESS_TYPES environment variable, or in the default address types mapping file if that environment variable is not set. This switch may not be combined with the **--map-file** or **--country-codes** switches.

--address-types=*ADDRTYPE_FILE*

Print the contents of the address types mapping file specified by *ADDRTYPE_FILE*.

--country-codes

Print the contents of the country code mapping file (**ccfilter(3)**) specified by the SILK_COUNTRY_CODES environment variable, or in the default country code mapping file if that environment variable is not set. This switch may not be combined with the **--map-file** or **--address-types** switches.

--country-codes=*COUNTRY_CODE_FILE*

Print the contents of the country code mapping file specified by *COUNTRY_CODE_FILE*.

--output-types={type | mapname | label | ranges}

Specify the type(s) of output to produce. When this switch is not provided, the default is to print **ranges**. Specify multiple types as a comma separated list of names; regardless of the order in which the types are given, the output will appear in the order shown below. Country-code prefix map files only support the `ranges` output type. A type can be specified using the shortest unique prefix for the type. The available types are:

type

Print the type of this prefix map file. The value will be one of `IPv4-address`, `IPv6-address`, or `proto-port`. The type will be preceded by the string `TYPE:` and a space character unless **--no-titles** is specified.

mapname

Print the name that is stored in the prefix map file. This mapname is used to generate switch names and field names when this prefix map is used with **rwfilter(1)**, **rwcut(1)**, **rwgroup(1)**, **rwsort(1)**, **rwstats(1)**, and **rwuniq(1)**. See **pmapfilter(3)** for details. The mapname will be preceded by the string `MAPNAME:` and a space character unless **--no-titles** is specified.

label

Print the names of the labels that exist in the prefix map file. The labels are printed left-justified, one per line, with no delimiter. The labels will be preceded by `LABELS:` on its own line unless **--no-titles** is specified. If **ranges** is also specified, a blank line will separate the labels and the range/label columns.

ranges

Print the range and label for each block in the prefix map file. If the prefix map contains protocol/port pairs, the output will contain three columns (startPair, endPair, label), where startPair and endPair contain *protocol/port*. If the prefix map contains IP addresses, the form of the output will depend on whether **--no-cidr-blocks** is specified. When it is not specified, the output will contain two columns (ipBlock, label), where ipBlock contains the IP range in CIDR notation. If **--no-cidr-blocks** is specified, the output will contain three columns: startIP, endIP, label.

--ignore-label=*LABEL*

For the **ranges** output-type, do not print entries whose label is *LABEL*. By default, all entries in the prefix map file are printed.

--ip-label-to-ignore=*IP_ADDRESS*

For the **ranges** output-type, find the label associated with the IP address *IP_ADDRESS* and ignore all ranges that match that label. By default, all entries in the prefix map are printed.

--left-justify-labels

For the **ranges** output-type, left-justify the labels when columnar output is printed. Normally, the labels are right-justified.

--no-cidr-blocks

Cause each IP address block to be printed as a starting and ending IP address. By default, IP addresses are grouped into CIDR blocks. This switch is ignored for prefix map files containing protocol/port pairs.

--ip-format=*FORMAT*

Specify how IP addresses are printed, where *FORMAT* is a comma-separated list of the arguments described below. This switch is ignored for prefix map files containing protocol/port pairs. When this switch is not specified, the SILK_IP_FORMAT environment variable is checked for a value and that format is used if it is valid. The default *FORMAT* is `canonical` according to whether the prefix map file is IPv4 or IPv6. *Since SiLK 3.7.0.*

canonical

Print IP addresses in the canonical format. For an IPv4 prefix map, use dot-separated decimal (`192.0.2.1`). For an IPv6 prefix map, use colon-separated hexadecimal (`2001:db8::1`) or a mixed IPv4-IPv6 representation for IPv4-mapped IPv6 addresses (the ::ffff:0:0/96 netblock, e.g., `::ffff:192.0.2.1`) and IPv4-compatible IPv6 addresses (the ::/96 netblock other than ::/127, e.g., `::192.0.2.1`).

no-mixed

Print IP addresses in the canonical format (`192.0.2.1` or `2001:db8::1`) but do not used the mixed IPv4-IPv6 representations. For example, use `::ffff:c000:201` instead of `::ffff:192.0.2.1`. *Since SiLK 3.17.0.*

decimal

Print IP addresses as integers in decimal format. For example, print `192.0.2.1` and `2001:db8::1` as `3221225985` and `42540766411282592856903984951653826561`, respectively.

hexadecimal

Print IP addresses as integers in hexadecimal format. For example, print `192.0.2.1` and `2001:db8::1` as `c00000201` and `20010db800000000000000000000001`, respectively. **Note:** This setting does not apply to CIDR prefix values which are printed as decimal.

zero-padded

Make all IP address strings contain the same number of characters by padding numbers with leading zeros. For example, print `192.0.2.1` and `2001:db8::1` as `192.000.002.001` and `2001:0db8:0000:0000:0000:0000:0000:0001`, respectively. For IPv6 addresses, this setting implies `no-mixed`, so that `::ffff:192.0.2.1` is printed as `0000:0000:0000:0000:0000:ffff:c000:0201`. As of SiLK 3.17.0, may be combined with any of the above, including `decimal` and `hexadecimal`. As of SiLK 3.18.0, the values of CIDR prefix are also zero-padded.

The following arguments modify certain IP addresses prior to printing. These arguments may be combined with the above formats.

map-v4

When the prefix map contains only IPv4 addresses, change all IPv4 addresses to IPv4-mapped IPv6 addresses (addresses in the ::ffff:0:0/96 netblock) prior to formatting. *Since SiLK 3.17.0.*

unmap-v6

When the prefix map contains IPv6 addresses, change any IPv4-mapped IPv6 addresses (addresses in the ::ffff:0:0/96 netblock) to IPv4 addresses prior to formatting. *Since SiLK 3.17.0.*

The following argument is also available:

force-ipv6

Set *FORMAT* to `map-v4`,`no-mixed`.

--integer-ips

Print IP addresses as integers. This switch is equivalent to **--ip-format=decimal**, it is deprecated as of SiLK 3.7.0, and it will be removed in the SiLK 4.0 release

--zero-pad-ips

Print IP addresses as fully-expanded, zero-padded values in their canonical form. This switch is equivalent to **--ip-format=zero-padded**, it is deprecated as of SiLK 3.7.0, and it will be removed in the SiLK 4.0 release.

--no-titles

Turn off column titles. By default, titles are printed.

--no-columns

Disable fixed-width columnar output.

--column-separator=*C*

Use specified character between columns and after the final column. When this switch is not specified, the default of '|' is used.

--no-final-delimiter

Do not print the column separator after the final column. Normally a delimiter is printed.

--delimited

--delimited=*C*

Run as if **--no-columns --no-final-delimiter --column-sep**=*C* had been specified. That is, disable fixed-width columnar output; if character *C* is provided, it is used as the delimiter between columns instead of the default '|'.

--output-path=*PATH*

Write the textual output to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output (and bypass the paging program). If *PATH* names an existing file, **rwpmapcat** exits with an error unless the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. If this switch is not given, the output is either sent to the pager or written to the standard output. *Since SiLK 3.15.0.*

--pager=*PAGER_PROG*

When output is to a terminal, invoke the program *PAGER_PROG* to view the output one screen full at a time. This switch overrides the SILK_PAGER environment variable, which in turn overrides the PAGER variable. If the **--output-path** switch is given or if the value of the pager is determined to be the empty string, no paging is performed and all output is written to the terminal.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

rwpmapbuild(1) creates the prefix map file *sample.pmap* from the textual input.

```
$ cat sample.txt
mode      ip
map-name  addrtype
label     0  non-routable
label     1  internal
label     2  external
default   external
         0.0.0.0/8  non-routable
        10.0.0.0/8  non-routable
       127.0.0.0/8  non-routable
    169.254.0.0/16  non-routable
     172.16.0.0/12  non-routable
      192.0.2.0/24  non-routable
    192.168.0.0/16  non-routable
255.255.255.255/32  non-routable
$ rwpmapbuild --input-path=sample.txt --output-path=sample.txt
```

Invoking **rwpmapcat** with the name of the file as its only argument prints the range-to-label contents of the prefix map file, and the contents are printed as CIDR blocks if the file contains IP addresses.

```
$ rwpmapcat sample.pmap | head -10
           ipBlock|       label|
         0.0.0.0/8|non-routable|
         1.0.0.0/8|    external|
         2.0.0.0/7|    external|
         4.0.0.0/6|    external|
         8.0.0.0/7|    external|
        10.0.0.0/8|non-routable|
        11.0.0.0/8|    external|
        12.0.0.0/6|    external|
        16.0.0.0/4|    external|
```

Use the **--no-cidr-blocks** switch to print the range as a pair of IPs. The **--map-file** switch may be use to specify the name of the file.

```
$ rwpmapcat --map-file=sample.pmap --no-cidr-block
        startIP|          endIP|       label|
        0.0.0.0|  0.255.255.255|non-routable|
        1.0.0.0|  9.255.255.255|    external|
       10.0.0.0| 10.255.255.255|non-routable|
       11.0.0.0|126.255.255.255|    external|
      127.0.0.0|127.255.255.255|non-routable|
      128.0.0.0|169.253.255.255|    external|
    169.254.0.0|169.254.255.255|non-routable|
    169.255.0.0| 172.15.255.255|    external|
     172.16.0.0| 172.31.255.255|non-routable|
     172.32.0.0|    192.0.1.255|    external|
      192.0.2.0|    192.0.2.255|non-routable|
      192.0.3.0|192.167.255.255|    external|
    192.168.0.0|192.168.255.255|non-routable|
    192.169.0.0|255.255.255.254|    external|
255.255.255.255|255.255.255.255|non-routable|
```

The **--output-types** switch determines what output is produced. Specifying an argument of `label` prints the labels that were specified when the file was built.

```
$ rwpmapcat --map-file=sample.pmap --output-types=label
LABELS:
non-routable
internal
external
```

Multiple types of output may be requested

```
$ rwpmapcat --map-file=sample.pmap --output-types=type,mapname
TYPE:  IPv4-address
MAPNAME:  addrtype
```

Sometimes the content of the prefix map more clear if you eliminate the ranges that were assigned to the default label. There are two ways to filter a label: either specify the label with the **--ignore-label** switch or find an IP address that has that label and specify the IP address to the **--ip-label-to-ignore** switch:

```
$ cat sample.pmap | rwpmapcat --ignore-label=external
           ipBlock|       label|
         0.0.0.0/8|non-routable|
        10.0.0.0/8|non-routable|
       127.0.0.0/8|non-routable|
    169.254.0.0/16|non-routable|
     172.16.0.0/12|non-routable|
      192.0.2.0/24|non-routable|
    192.168.0.0/16|non-routable|
255.255.255.255/32|non-routable|

$ cat sample.pmap | rwpmapcat --ip-label-to-ignore=0.0.0.0 | head -7
           ipBlock|       label|
         1.0.0.0/8|    external|
         2.0.0.0/7|    external|
         4.0.0.0/6|    external|
         8.0.0.0/7|    external|
        11.0.0.0/8|    external|
        12.0.0.0/6|    external|
```

rwpmapcat also supports viewing the contents of prefix map files containing protocol/port pairs.

```
$ rwpmapcat proto.pmap
startPair|  endPair|                          label|
...
      6/0|      6/0|                            TCP|
      6/1|      6/1|                          tcpmux|
      6/2|      6/3|                     compressnet|
      6/4|      6/4|                            TCP|
      6/5|      6/5|                            rje|
      6/6|      6/6|                            TCP|
      6/7|      6/7|                           echo|
      6/8|      6/8|                            TCP|
...
```

As of SiLK 3.8.0, **rwpmapcat** supports printing the contents of the country code mapping file created by **rwgeoip2ccmap(1)** (for use in the country code plug-in **ccfilter(3)**) when the **--country-codes** switch is used.

```
$ rwpmapcat --no-cidr --country-codes=country_codes.pmap | head
        startIP|          endIP|label|
         0.0.0.0|     2.6.190.55|   --|
     2.6.190.56|     2.6.190.63|   gb|
     2.6.190.64|  2.255.255.255|   --|
        3.0.0.0|    4.17.135.31|   us|
    4.17.135.32|    4.17.135.63|   ca|
    4.17.135.64|   4.17.142.255|   us|
     4.17.143.0|    4.17.143.15|   ca|
    4.17.143.16|     4.18.32.71|   us|
     4.18.32.72|     4.18.32.79|   mx|
```

ENVIRONMENT

SILK_IP_FORMAT

This environment variable is used as the value for **--ip-format** when that switch is not provided. *Since SiLK 3.11.0.*

SILK_PAGER

When set to a non-empty string, **rwpmapcat** automatically invokes this program to display its output a screen at a time. If set to an empty string, **rwpmapcat** does not automatically page its output.

PAGER

When set and SILK_PAGER is not set, **rwpmapcat** automatically invokes this program to display its output a screen at a time.

SILK_COUNTRY_CODES

This environment variable allows the user to specify the country code mapping file to use when the **--country-codes** switch is specified without an argument. The variable's value may be a complete path or a file relative to SILK_PATH. See the FILES section for standard locations of this file.

SILK_ADDRESS_TYPES

This environment variable allows the user to specify the address type mapping file to use when the **--address-types** switch is specified without an argument. The variable's value may be a complete path or a file relative to the SILK_PATH. See the FILES section for standard locations of this file.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **rwpmapcat** may use this environment variable. See the FILES section for details.

FILES

${SILK_COUNTRY_CODES}

${SILK_PATH}/share/silk/country_codes.pmap

${SILK_PATH}/share/country_codes.pmap

/usr/local/share/silk/country_codes.pmap

/usr/local/share/country_codes.pmap

Possible locations for the country codes mapping file when the **--country-codes** switch is specified without an argument.

${SILK_ADDRESS_TYPES}

${SILK_PATH}/share/silk/address_types.pmap

${SILK_PATH}/share/address_types.pmap

/usr/local/share/silk/address_types.pmap

/usr/local/share/address_types.pmap

Possible locations for the address types mapping file when the **--address-types** switch is specified without an argument.

SEE ALSO

rwpmapbuild(1), **rwgeoip2ccmap(1)**, **addrtype(3)**, **pmapfilter(3)**, **ccfilter(3)**, **rwfilter(1)**, **rwcut(1)**, **rwgroup(1)**, **rwsort(1)**, **rwstats(1)**, **rwuniq(1)**, **silk(7)**

NOTES

The **--country-codes** and **--address-types** switches were added in SiLK 3.8.0.

rwpmaplookup

Map keys to prefix map entries

SYNOPSIS

```
rwpmaplookup { --map-file=MAP_FILE | --address-types[=MAP_FILE]
               | --country-codes[=MAP_FILE] }
      [--fields=FIELDS] [--ipset-files] [--no-errors]
      [--ip-format=FORMAT] [--integer-ips] [--zero-pad-ips]
      [--no-titles] [--no-columns] [--column-separator=CHAR]
      [--no-final-delimiter] [{--delimited | --delimited=CHAR}]
      [{--output-path=PATH | --pager=PAGER_PROG}]
      [--no-files ARG [ARGS...] | --xargs[=FILE] | FILE [FILES...]]

rwpmaplookup --help

rwpmaplookup --version
```

DESCRIPTION

rwpmaplookup finds keys in a binary prefix map file and prints the key and its value in a textual, bar (|) delimited format.

By default, **rwpmaplookup** expects its arguments to be the names of text files containing keys---one key per line. When the **--ipset-files** switch is given, **rwpmaplookup** takes IPset files as arguments and uses the IPs as the keys. The **--no-files** switch causes **rwpmaplookup** to treat each command line argument itself as a key to find in the prefix map.

When **--no-files** is not specified, **rwpmaplookup** reads the keys from the files named on the command line or from the standard input when no file names are specified and neither **--xargs** nor **--no-files** is present. To read the standard input in addition to the named files, use `-` or `stdin` as a file name. When the **--xargs** switch is provided, **rwpmaplookup** reads the names of the files to process from the named text file or from the standard input if no file name argument is provided to the switch. The input to **--xargs** must contain one file name per line.

You must tell **rwpmaplookup** the prefix map to use for look-ups using one of three switches:

- To use an arbitrary prefix map, use the **--map-file** switch.
- If you want to map IP addresses to country codes (see **ccfilter(3)**), use the **--country-codes** switch. To use the default country code prefix map, do not provide an argument to the switch. To use a specific country code mapping file, specify the file as the argument.
- If you want to map IP addresses to address types (see **addrtype(3)**), use the **--address-types** switch. To use the default address types prefix map, do not provide an argument to the switch. To use a specific address types mapping file, specify the file as the argument.

If the **--map-file** switch specifies a prefix map containing protocol/port pairs, each input file should contain one protocol/port pair per line in the form *PROTOCOL*/*PORT*, where *PROTOCOL* is a number between

0 and 255 inclusive, and *PORT* is a number between 0 and 65535 inclusive. When the **--ipset-files** switch is specified, it is an error if the **--map-file** switch specifies a prefix map containing protocol/port pairs.

When querying any other type of prefix map and the **--ipset-files** switch is not present, each textual input file should contain one IP address per line, where the IP is a single IP address (*not* a CIDR block) in canonical form or the integer representation of an IPv4 address.

The **--fields** switch allows you to specify which columns appear in the output. The default columns are the key and the value, where the key is the IP address or protocol/port pair, and the value is the textual label for that key.

If the prefix map contains IPv6 addresses, any IPv4 address in the input is mapped into the ::ffff:0:0/96 netblock when searching.

If the prefix map contains IPv4 addresses only, any IPv6 address in the ::ffff:0:0/96 netblock is converted to IPv4 when searching. Any other IPv6 address is ignored, and it is not printed in the output unless the `input` field is requested.

Prefix map files are created by the **rwpmapbuild(1)** and **rwgeoip2ccmap(1)** utilities. IPset files are created most often by **rwset(1)** and **rwsetbuild(1)**.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

One of **--map-file**, **--address-types**, or **--country-codes** is required.

--map-file=*PMAP_FILE*

Find the IP addresses or protocol/port pairs in the prefix map file *PMAP_FILE*.

--address-types

Find the IP addresses in the address types (see **addrtype(3)**) mapping file specified by the SILK_ADDRESS_TYPES environment variable, or in the default address types mapping file if that environment variable is not set.

--address-types=*ADDRTYPE_FILE*

Find the IP addresses in the address types mapping file specified by *ADDRTYPE_FILE*.

--country-codes

Find the IP addresses in the country code (see **ccfilter(3)**) mapping file specified by the SILK_COUNTRY_CODES environment variable, or in the default country code mapping file if that environment variable is not set.

--country-codes=*COUNTRY_CODE_FILE*

Find the IP addresses in the country code mapping file specified by *COUNTRY_CODE_FILE*.

--fields=*FIELDS*

Specify the columns to include in the output. The columns are displayed in the order the fields are specified. *FIELDS* is a comma separated list of field-names. Field-names are case-insensitive. When this switch is not provided, the default fields are `key,value`. The list of available fields are:

key
The key used to search the prefix map.

value
The label returned from the prefix map for the key.

block
The block in the prefix map that contains the key. For a prefix map file that contains IPv4 addresses, the result will be a CIDR block such as 10.18.26.32/27.

start-block
The value at the start of the block in the prefix map that contains the key.

end-block
The value at the end of the block in the prefix map that contains the key.

input
The text read from the input file that **rwpmaplookup** attempted to parse. Note that blank lines, lines containing only whitespace and comments, and lines longer than 2048 characters will not be printed. In addition, any comments appearing after the text are stripped. When **--ipset-files** is specified, this field contains the IP address in its canonical form.

--no-files

Causes **rwpmaplookup** to treat the command line arguments as the text to be parsed. This allows one to look up a handful of values without having to create a temporary file. Use of the **--no-files** switch disables paging of the output. This switch may not be combined with **--ipset-files**.

--no-errors

Disables printing of errors when the input cannot be parsed as an IP address or a protocol/port pair. This switch is ignored when **--ipset-files** is specified.

--ipset-files

Causes **rwpmaplookup** to treat the command line arguments as the names of IPset files to read and use as keys into the prefix map. It is an error to use this switch when **--map-file** specifies a protocol/port prefix map. When **--ipset-files** is active, the `input` column of **--fields** contains the IP in its canonical form, regardless of the **--ip-format** switch. This switch may not be combined with **--no-files**.

--ip-format=*FORMAT*

When printing the key of a prefix map containing IP addresses, specify how IP addresses are printed, where *FORMAT* is a comma-separated list of the arguments described below. When this switch is not specified, the SILK_IP_FORMAT environment variable is checked for a value and that format is used if it is valid. The default *FORMAT* is `canonical` according to whether the prefix map file is IPv4 or IPv6. *Since SiLK 3.7.0.*

canonical
Print IP addresses in the canonical format. For an IPv4 prefix map, use dot-separated decimal (`192.0.2.1`). For an IPv6 prefix map, use colon-separated hexadecimal (`2001:db8::1`) or a mixed IPv4-IPv6 representation for IPv4-mapped IPv6 addresses (the ::ffff:0:0/96 netblock, e.g., `::ffff:192.0.2.1`) and IPv4-compatible IPv6 addresses (the ::/96 netblock other than ::/127, e.g., `::192.0.2.1`).

no-mixed
Print IP addresses in the canonical format (`192.0.2.1` or `2001:db8::1`) but do not used the mixed IPv4-IPv6 representations. For example, use `::ffff:c000:201` instead of `::ffff:192.0.2.1`. *Since SiLK 3.17.0.*

decimal
Print IP addresses as integers in decimal format. For example, print `192.0.2.1` and `2001:db8::1` as `3221225985` and `42540766411282592856903984951653826561`, respectively.

hexadecimal
Print IP addresses as integers in hexadecimal format. For example, print `192.0.2.1` and `2001:db8::1` as `c00000201` and `20010db8000000000000000000000001`, respectively. **Note:** This setting does not apply to CIDR prefix values which are printed as decimal.

zero-padded
Make all IP address strings contain the same number of characters by padding numbers with leading zeros. For example, print `192.0.2.1` and `2001:db8::1` as `192.000.002.001` and `2001:0db8:0000:0000:0000:0000:0000:0001`, respectively. For IPv6 addresses, this setting implies `no-mixed`, so that `::ffff:192.0.2.1` is printed as `0000:0000:0000:0000:0000:ffff:c000:0201`. As of SiLK 3.17.0, may be combined with any of the above, including `decimal` and `hexadecimal`. As of SiLK 3.18.0, the values of CIDR prefix are also zero-padded.

The following arguments modify certain IP addresses prior to printing. These arguments may be combined with the above formats.

map-v4
When the prefix map contains only IPv4 addresses, change all IPv4 addresses to IPv4-mapped IPv6 addresses (addresses in the ::ffff:0:0/96 netblock) prior to formatting. *Since SiLK 3.17.0.*

unmap-v6
When the prefix map contains IPv6 addresses, change any IPv4-mapped IPv6 addresses (addresses in the ::ffff:0:0/96 netblock) to IPv4 addresses prior to formatting. *Since SiLK 3.17.0.*

The following argument is also available:

force-ipv6
Set *FORMAT* to `map-v4`,`no-mixed`.

--integer-ips
Print IP addresses as integers. This switch is equivalent to **--ip-format=decimal**, it is deprecated as of SiLK 3.7.0, and it will be removed in the SiLK 4.0 release.

--zero-pad-ips
Print IP addresses as fully-expanded, zero-padded values in their canonical form. This switch is equivalent to **--ip-format=zero-padded**, it is deprecated as of SiLK 3.7.0, and it will be removed in the SiLK 4.0 release.

--no-titles
Turn off column titles. By default, titles are printed.

--no-columns
Disable fixed-width columnar output.

--column-separator=*C*
Use specified character between columns and after the final column. When this switch is not specified, the default of '|' is used.

--no-final-delimiter
Do not print the column separator after the final column. Normally a delimiter is printed.

--delimited

--delimited=*C*

Run as if **--no-columns --no-final-delimiter --column-sep**=*C* had been specified. That is, disable fixed-width columnar output; if character *C* is provided, it is used as the delimiter between columns instead of the default '|'.

--output-path=*PATH*

Write the textual output to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output (and bypass the paging program). If *PATH* names an existing file, **rwpmaplookup** exits with an error unless the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. If this option is not given, the output is either sent to the pager or written to the standard output.

--pager=*PAGER_PROG*

When the **--no-files** switch has not been specified and output is to a terminal, invoke the program *PAGER_PROG* to view the output one screen full at a time. This switch overrides the SILK_PAGER environment variable, which in turn overrides the PAGER variable. If the **--output-path** switch is given or if the value of the pager is determined to be the empty string, no paging is performed and all output is written to the terminal.

--xargs

--xargs=*FILENAME*

Read the names of the input files from *FILENAME* or from the standard input if *FILENAME* is not provided. The input is expected to have one filename per line. **rwpmaplookup** opens each named file in turn and reads the IPset, the textual IP addresses, or the textual protocol/port pairs from it as if the filenames had been listed on the command line.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

Country code examples

Print the country code for a list of addresses read from the standard input.

```
$ cat my-addrs.txt
128.2.0.0
128.2.0.1
$ cat my-addrs.txt | rwpmaplookup --country-codes
            key|               value|
      128.2.0.0|                  us|
      128.2.0.1|                  us|
```

Use **--no-files** to list the address on the command line.

```
$ rwpmaplookup --country-codes  128.2.0.0 128.2.0.1
            key|               value|
      128.2.0.0|                  us|
      128.2.0.1|                  us|
```

Use **--ipset-files** to read the addresses from an IPset file.

```
$ rwsetbuild my-addrs.txt my-addrs.set
$ rwpmaplookup --country-codes --ipset-files my-addrs.set
            key|               value|
      128.2.0.0|                  us|
      128.2.0.1|                  us|
```

Use the **--fields** switch to control which columns are printed.

```
$ rwpmaplookup --country-codes --fields=value my-addrs.txt
               value|
                  us|
                  us|
```

Add the **--delimited** and **--no-titles** switches so the output only contains the value column. Print the country code for a single address using the default country code prefix map.

```
$ rwpmaplookup --country-codes --fields=value --delimited \
       --no-titles --no-files 128.2.0.0
us
```

Alternatively

```
$ echo 128.2.0.0   \
  | rwpmaplookup --country-codes --fields=value --delim --no-title
us
```

To use a different country code mapping file, provide that file as the argument to the **--country-codes** switch.

```
$ rwpmaplookup --country-code=old-address-map.pmap --no-files 128.2.0.0
          key|value|
    128.2.0.0|   us|
```

CIDR block input

Note that **rwpmaplookup** does not parse text that contains CIDR blocks.

```
$ echo '128.2.0.0/31'      \
  | rwpmaplookup --country-codes
            key|value|
rwpmaplookup: Invalid IP '128.2.0.1/31' at -:1: Extra text follows value
```

For this case, use the IPset tool **rwsetbuild(1)** to parse the CIDR block list and create a binary IPset stream, and pipe the IPset to **rwpmaplookup**.

```
$ echo '128.2.0.0/31'      \
  | rwsetbuild             \
  | rwpmaplookup --country-code --ipset-files
           key|value|
      128.2.0.0|   --|
      128.2.0.1|   --|
```

For versions of **rwpmaplookup** that do not have the **--ipset-files** switch, you can have **rwsetcat(1)** read the binary IPset stream and print the IP addresses as text, and pipe that into **rwpmaplookup**. Be sure to include the `--cidr-blocks=0` switch to **rwsetcat** which forces individual IP addresses to be printed.

```
$ echo '128.2.0.0/31'              \
  | rwsetbuild                     \
  | rwsetcat --cidr-blocks=0       \
  | rwpmaplookup --country-code
           key|value|
      128.2.0.0|   --|
      128.2.0.1|   --|
```

General prefix map usage

Consider a user-defined prefix map, *assigned-slash-8s.pmap*, that maps each /8 in the IPv4 address space to its assignment.

```
$ rwpmapcat assigned-slash-8s.pmap | head -4
          ipBlock|                                          label|
        0.0.0.0/8|                  IANA - Local Identification|
        1.0.0.0/8|                                          APNIC|
        2.0.0.0/8|                                       RIPE NCC|
```

Use the **--map-file** switch to map from IPs to labels using this prefix map.

```
$ cat my-addrs.txt
17.17.17.17
9.9.9.9
$ cat my-addrs.txt | rwpmaplookup --map-file=assigned-slash-8s.pmap
            key|               value|
    17.17.17.17| Apple Computer Inc.|
        9.9.9.9|                IBM|
```

Use **--ip-format=decimal** to print the output as integers.

```
$ cat my-addrs.txt         \
  | rwpmaplookup --ip-format=decimal --map-file=assigned-slash-8s.pmap
       key|               value|
 286331153| Apple Computer Inc.|
 151587081|                IBM|
```

Add the `input` field to see the input as well.

```
$ cat my-addrs.txt         \
  | rwpmaplookup --ip-format=decimal --fields=key,value,input \
       --map-file=assigned-slash-8s.pmap
       key|               value|               input|
 286331153| Apple Computer Inc.|         17.17.17.17|
 151587081|                 IBM|             9.9.9.9|
```

Combine the `input` field with the **--no-errors** switch to see a row for each key.

```
$ rwpmaplookup --fields=key,value,input --no-errors --no-files \
       --map-file=assigned-slash-8s.pmap 9.9.9.9 17.1717.17
           key|               value|               input|
        9.9.9.9| Apple Computer Inc.|             9.9.9.9|
               |                    |          17.1717.17|
```

The input can contain integer values.

```
$ echo 151587081           \
  | rwpmaplookup --fields=key,value,input --delimited=, \
       --map-file=assigned-slash-8s.pmap
key,value,input
9.9.9.9,IBM,151587081
```

Block output

Specifying `block` in the **--fields** switch causes **rwpmaplookup** to print the CIDR block that contains the address key.

```
$ cat my-addrs.txt
9.8.7.6
9.10.11.12
17.16.15.14
17.18.19.20
$ rwpmaplookup --map-file=assigned-slash-8s.pmap \
       --fields=key,value,block my-addrs.txt
           key|               value|             block|
        9.8.7.6|                 IBM|         9.0.0.0/8|
     9.10.11.12|                 IBM|         9.0.0.0/8|
    17.16.15.14| Apple Computer Inc.|        17.0.0.0/8|
    17.18.19.20| Apple Computer Inc.|        17.0.0.0/8|
```

To break the CIDR block into its starting and ending value, specify the `start-block` and `end-block` fields.

```
$ rwpmaplookup --map-file=assigned-slash-8s.pmap              \
       --fields=key,value,start-block,end-block my-addrs.txt
           key|               value|    start-block|      end-block|
        9.8.7.6|                 IBM|        9.0.0.0|  9.255.255.255|
```

```
     9.10.11.12|                IBM|        9.0.0.0|  9.255.255.255|
    17.16.15.14| Apple Computer Inc.|       17.0.0.0| 17.255.255.255|
    17.18.19.20| Apple Computer Inc.|       17.0.0.0| 17.255.255.255|
```

To get a unique list of blocks for the input keys, do not output the `key` field and pipe the output of **rwpmaplookup** to the **uniq(1)** command. (This works as long as the input data is sorted).

```
$ cat my-addrs.txt                                  \
  | rwpmaplookup --map-file=assigned-slash-8s.pmap \
      --fields=block,value                          \
  | uniq
            block|               value|
         9.0.0.0/8|                 IBM|
        17.0.0.0/8| Apple Computer Inc.|
```

The values printed in the `block` column corresponds to the CIDR block that were used when the prefix map file was created.

```
$ rwpmaplookup --map=assigned-slash-8s.pmap --fields=block,value   \
       --no-files 128.2.0.1 129.0.0.1
            block|               value|
       128.0.0.0/8|Administered by ARIN|
       129.0.0.0/8|Administered by ARIN|
```

In the output from **rwpmapcat(1)**, those two blocks are combined into a larger range.

```
$ rwpmapcat --map=assigned-slash-8s.pmap | grep 128
       128.0.0.0/6|Administered by ARIN|
```

Working with IPsets

Assume you have a binary IPset file, *my-ips.set*, that has the contents shown here, and you want to find the list of unique assignments from the *assigned-slash-8s.pmap* file.

```
$ rwsetcat --cidr-blocks=1 my-ips.set
9.9.9.0/24
13.13.13.0/24
15.15.15.0/24
16.16.16.0/24
17.17.17.0/24
18.18.18.0/24
```

Since the blocks in the *assigned-slash-8s.pmap* file are /8, use the **rwsettool(1)** command to mask the IPs in the IPset to the unique /8 that contains each of the IPs.

```
$ rwsettool --mask=8 my-ips.set    \
  | rwpmaplookup --map-file=assigned-slash-8s.pmap
          key|                        value|
      9.0.0.0|                          IBM|
```

```
      13.0.0.0|            Xerox Corporation|
      15.0.0.0|      Hewlett-Packard Company|
      16.0.0.0|Digital Equipment Corporation|
      17.0.0.0|          Apple Computer Inc.|
      18.0.0.0|                           MIT|
```

Protocol/port prefix maps

Assume the **service.pmap** prefix map file maps protocol/port pairs to the name of the service running on the named port.

```
$ rwpmapcat service.pmap
startPair|  endPair|    label|
      0/0|  0/65535|  unknown|
      1/0|  1/65535|     ICMP|
      2/0|  5/65535|  unknown|
      6/0|     6/21|      TCP|
     6/22|     6/22|  TCP/SSH|
...
     17/0|    17/52|      UDP|
    17/53|    17/53|  UDP/DNS|
...
```

To query this prefix map, the input must contain two numbers separated by a slash.

```
$ rwpmaplookup --map-file=service.pmap --no-files 6/80
      key|    value|
     6/80| TCP/HTTP|
```

Specifying `block`, `start-block`, and `end-block` in the **--fields** switch also works for Protocol/port prefix map files. The `block` column contains the same information as the `start-block` and `end-block` columns separated by a single space.

```
$ rwpmaplookup --map-file=service.pmap --no-files  \
       --fields=key,value,start,end,block          \
       6/80 6/6000 17/0 17/53 128/128
      key|     value|start-blo|end-block|             block|
     6/80|  TCP/HTTP|     6/80|     6/80|          6/80 6/80|
   6/6000|       TCP|   6/4096|   6/6143|      6/4096 6/6143|
     17/0|       UDP|     17/0|    17/31|         17/0 17/31|
    17/53|   UDP/DNS|    17/53|    17/53|        17/53 17/53|
  200/200|Unassigned|    192/0|223/65535|    192/0 223/65535|
```

Using the **pmapfilter(3)** plug-in to **rwcut(1)**, you can print the label for the source port and destination port in the SiLK Flow file *data.rw*.

```
$ rwcut --pmap-file=service.pmap --num-rec=5       \
       --fields=proto,sport,src-service,dport,dst-service data.rw
pro|sPort|src-service|dPort|dst-service|
```

```
 17|29617|        UDP|   53|    UDP/DNS|
 17|   53|    UDP/DNS|29617|        UDP|
  6|29618|        TCP|   22|    TCP/SSH|
  6|   22|    TCP/SSH|29618|        TCP|
  1|    0|       ICMP|  771|       ICMP|
```

The **pmapfilter** plug-in does not provide a way to print the values based on the application field. You can get that information by having **rwcut** print the protocol and application separated by a slash, and pipe the result into **rwpmaplookup**.

```
$ rwcut --fields=proto,application --num-rec=5     \
       --delimited=/ --no-title                    \
  | rwpmaplookup --map-file=service.pmap
      key|    value|
    17/53|  UDP/DNS|
    17/53|  UDP/DNS|
     6/22|  TCP/SSH|
     6/22|  TCP/SSH|
      1/0|     ICMP|
```

ENVIRONMENT

SILK_IP_FORMAT

This environment variable is used as the value for **--ip-format** when that switch is not provided. *Since SiLK 3.11.0.*

SILK_PAGER

When set to a non-empty string, **rwpmaplookup** automatically invokes this program to display its output a screen at a time unless the **--no-files** switch is given. If this variable is set to an empty string, **rwpmaplookup** does not automatically page its output.

PAGER

When set and SILK_PAGER is not set, **rwpmaplookup** automatically invokes this program to display its output a screen at a time.

SILK_COUNTRY_CODES

This environment variable allows the user to specify the country code mapping file to use when the **--country-codes** switch is specified without an argument. The variable's value may be a complete path or a file relative to SILK_PATH. See the FILES section for standard locations of this file.

SILK_ADDRESS_TYPES

This environment variable allows the user to specify the address type mapping file to use when the **--address-types** switch is specified without an argument. The variable's value may be a complete path or a file relative to the SILK_PATH. See the FILES section for standard locations of this file.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **rwpmaplookup** may use this environment variable. See the FILES section for details.

FILES

${SILK_COUNTRY_CODES}

${SILK_PATH}/share/silk/country_codes.pmap

${SILK_PATH}/share/country_codes.pmap

/usr/local/share/silk/country_codes.pmap

/usr/local/share/country_codes.pmap

Possible locations for the country codes mapping file when the **--country-codes** switch is specified without an argument.

${SILK_ADDRESS_TYPES}

${SILK_PATH}/share/silk/address_types.pmap

${SILK_PATH}/share/address_types.pmap

/usr/local/share/silk/address_types.pmap

/usr/local/share/address_types.pmap

Possible locations for the address types mapping file when the **--address-types** switch is specified without an argument.

NOTES

rwpmaplookup was added in SiLK 3.0.

rwpmaplookup duplicates the functionality of **rwip2cc(1)**. **rwip2cc** is deprecated, and it will be removed in the SiLK 4.0 release. Examples of using **rwpmaplookup** in place of **rwip2cc** are provided in the latter's manual page.

SEE ALSO

rwpmapbuild(1), **rwpmapcat(1)**, **ccfilter(3)**, **addrtype(3)**, **pmapfilter(3)**, **rwgeoip2ccmap(1)**, **rwcut(1)**, **rwset(1)**, **rwsetbuild(1)**, **rwsetcat(1)**, **rwsettool(1)**, **silk(7)**, **uniq(1)**

rwpmatch

Filter a tcpdump file using a SiLK Flow file

SYNOPSIS

```
rwpmatch --flow-file=FLOW_FILE [--msec-compare] [--ports-compare]
      TCPDUMP_INPUT > TCPDUMP_OUTPUT

rwpmatch --help

rwpmatch --version
```

DESCRIPTION

rwpmatch reads each packet from the **pcap(3)** (**tcpdump(1)**) capture file *TCPDUMP_INPUT* and writes the packet to the standard output if the specified *FLOW_FILE* contains a matching SiLK Flow record. It is designed to reverse the input from **rwptoflow(1)**.

rwpmatch will read the pcap capture data from its standard input if *TCPDUMP_INPUT* is specified as `stdin`. The application will fail when attempting to read or write binary data from or to a terminal.

The SiLK Flow records in *FLOW_FILE* should appear in time sorted order.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--flow-file=*FLOW_FILE*

FLOW_FILE refers to a file, named pipe, or the string `stdin`. The flow file determines which packet records should be output to the new packet file. This switch is required.

--msec-compare

Compare times down to the millisecond (rather than the default of second).

--ports-compare

For TCP and UDP data, compare the source and destination ports when matching.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line.

Given the **pcap** capture file *data.pcap*, use **rwptoflow(1)** to convert it to a SiLK flow file:

```
$ rwptoflow data.pcap --packet-pass=good.pcap --flow-out=data.rw
```

With **rwfilter(1)**, select the SiLK Flow records whose source IPs are found in the IPset file *sip.set*:

```
$ rwfilter --sipset=sip.set --pass=filtered.rw  data.rw
```

Match the original **pcap** file against the filtered SiLK file, in effect generating a **pcap** file which has been filtered by *sip.set*:

```
$ rwpmatch --flow-file=filtered.rw good.pcap > filtered.pcap
```

NOTES

For best results, the **tcpdump** input to **rwpmatch** should be the output from **--packet-pass-output** switch on **rwptoflow**. This ensures that only well-behaved packets are given to **rwpmatch**.

The flow file input to **rwpmatch** should contain single-packet flows originally derived from a **tcpdump** file using **rwptoflow**. If a flow record is found which does not represent a corresponding **tcpdump** record, **rwpmatch** will return an error.

Both the **tcpdump** and the SiLK file inputs must be time-ordered.

rwpmatch is an expensive I/O application since it reads the entire **tcpdump** capture file and the entire SiLK Flow file. It may be worthwhile to optimize an analysis process to avoid using **rwpmatch** until payload filtering is necessary. Saving the output from **rwpmatch** as a partial-results file, and matching against that in the future (rather than the original **tcpdump** file) can also provide significant performance gains.

SiLK supports millisecond timestamps. When reading packets whose timestamps have finer precision, the times are truncated at the millisecond position.

SEE ALSO

rwptoflow(1), **rwfilter(1)**, **silk(7)**, **tcpdump(1)**, **pcap(3)**

rwptoflow

Generate SiLK Flow records from packet data

SYNOPSIS

```
rwptoflow [--plugin=PLUGIN [--plugin=PLUGIN ...]]
      [--active-time=YYYY/MM/DD:hh:dd:mm:ss.uuuuuu-YYYY/MM/DD:hh:dd:mm:ss.uuuuuu]
      [--flow-output=FLOW_PATH] [--packet-pass-output=PCKTS_PASS]
      [--packet-reject-output=PCKTS_REJECT]
      [--reject-all-fragments] [--reject-nonzero-fragments]
      [--reject-incomplete] [--set-sensorid=SCALAR]
      [--set-inputindex=SCALAR] [--set-outputindex=SCALAR]
      [--set-nexthopip=IP_ADDRESS] [--print-statistics]
      [--note-add=TEXT] [--note-file-add=FILE]
      [--compression-method=COMP_METHOD] TCPDUMP_INPUT

rwptoflow [--plugin=PLUGIN ...] --help

rwptoflow --version
```

DESCRIPTION

rwptoflow attempts to generate a SiLK Flow record for every Ethernet IP IPv4 packet in the **pcap(3)** (**tcpdump(1)**) capture file *TCPDUMP_INPUT*. *TCPDUMP_INPUT* must contain data captured from an Ethernet datalink.

rwptoflow does not attempt to reassemble fragmented packets or to combine multiple packets into a single flow record. **rwptoflow** is a simple program that creates one SiLK Flow record for every IPv4 packet in *TCPDUMP_INPUT*. (For an alternate approach, consider using the **rwp2yaf2silk(1)** tool as described at the end of this section.)

rwptoflow will read from its standard input if *TCPDUMP_INPUT* is specified as `stdin`. The SiLK Flow records are written to the specified **flow-output** file or to the standard output. The application will fail when attempting to read or write binary data from or to a terminal.

Packets outside of a user-specified **active-time** window can be ignored. Additional filtering on the *TCPDUMP_INPUT* can be performed by using **tcpdump** with an `expression` filter and piping **tcpdump**'s output into **rwptoflow**.

In addition to generating flow records, **rwptoflow** can write pcap files containing the packets that it used to generate each flow, and/or the packets that were rejected. Note that packets falling outside the **active-time** window are ignored and are not written to the **packet-reject-output**.

Statistics of the number of packets read, rejected, and written can be printed.

rwptoflow will reject any packet that is not an IPv4 Ethernet packet and any packet that is too short to contain the Ethernet and IP headers. At the user's request, packets may be rejected when

- they are fragmented---either the initial (zero-offset) fragment or a subsequent fragment
- they have a non-zero fragment offset

- they are not fragmented or they are the zero-fragment but the capture file does not contain enough information about the packet to set protocol-specific information---namely the ICMP type and code, the UDP source and destination ports, or the TCP source and destination ports and flags

Since the input packet formats do not contain some fields normally found in NetFlow data, **rwptoflow** provides a way to set those flow values in all packets. For example, it is possible to set the sensor-id manually for a **tcpdump** source, so that flow data can be filtered or sorted by that value later.

Alternative to rwptoflow

As mentioned above, **rwptoflow** is a simple program for processing Ethernet IP IPv4 packets. **rwptoflow** does not:

- reassemble fragmented packets
- support IPv6 packets
- combine multiple packets into a single flow record
- support any decoding of packets (e.g., 802.1q)

For these features (and others), you should use the **yaf(1)** application (http://tools.netsa.cert.org/yaf/) to read the pcap file and generate an IPFIX stream, and pipe the IPFIX stream into **rwipfix2silk(1)** to convert it to SiLK Flow records.

The **rwp2yaf2silk(1)** script makes this common usage more convenient by wrapping the invocation of **yaf** and **rwipfix2silk**. You give **rwp2yaf2silk** a pcap file and it writes SiLK Flow records.

By default, **rwptoflow** creates a flow record for *every* packet, fragments and all. You can almost force **yaf** to create a flow record for every packet: When you give **yaf** the **--idle-timeout=0** switch, **yaf** creates a flow record for every complete packet and for each packet that it is able to completely reassemble from packet fragments. Any fragmented packets that **yaf** cannot reassemble are dropped.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--plugin=*PLUGIN*

Use the specified plug-in to ignore or reject packets or to modify the flow record that is generated from the packet. The switch may be repeated to load multiple plug-ins. See the PLUG-IN SUPPORT section below for details.

--active-time=*YYYY/MM/DD[:hh[:dd[:mm[:ss[.uuuuuu]]]]]*

--active-time=*YYYY/MM/DD[:hh[:dd[:mm[:ss[.uuuuuu]]]]]*-*YYYY/MM/DD[:hh[:dd[:mm[:ss[.uuuuuu]]]]]*

Ignore all packets whose time falls outside the specified range. The times must be specified to at least day precision. The start time is required; when the end-time is not present, it is treated as infinite. The end-time will be rounded-up to instant before the next time unit; i.e., an end-time of 2006/08/31:15 is treated as 2006/08/31:15:59:59.999999.

--flow-output=*FLOW_PATH*

Write the generated SiLK Flow records to the specified file at *FLOW_PATH*. When this switch is not provided, the flows are written to the standard output.

--packet-pass-output=*PCKTS_PASS*

For each generated SiLK Flow record, write the packet that generated the flow to the pcap file specified by *PCKTS_PASS*. Use `stdout` to write the packets to the standard output.

--packet-reject-output=*PCKTS_REJECT*

Write each packet that occurs within the active-time window but for which a SiLK Flow record was **not** generated to the pcap file specified by *PCKTS_REJECT*. Use `stdout` to write the packets to the standard output.

The packets that get written to this file may include packets that were shorter than that required to get the IP header, non-IPv4 packets, and packets that get treated as `reject` packets by the following switches.

--reject-all-fragments

Do not generate a SiLK Flow record for the packet when the packet is fragmented. This includes the initial (zero-offset) fragment and all subsequent fragments. If **--packet-reject-output** is specified, the packet will be written to that file.

--reject-nonzero-fragments

Do not generate a SiLK Flow record for the packet when the packet is fragmented **unless** this is the initial fragment. That is, reject all packets that have a non-zero fragmentation offset. Normally flow records are generated for these packets, but the ports and TCP flag information is set to zero. If **--packet-reject-output** is specified, the packet will be written to that file.

--reject-incomplete

Do not generate a SiLK Flow record for the packet when the packet's fragmentation-offset is zero yet the packet does not contain enough information to completely specify an ICMP, UDP, or TCP record (that is, the packet is too short to set the ICMP type and code, the UDP or TCP source or destination port, or the TCP flags). Normally, flow records are generated for these packets but the ports and TCP flag information is set to zero. This switch has no effect on packets where the protocol is not 1,6, or 17.

This switch does **not** imply **--reject-nonzero-fragments**; to indicate that *all* generated flow records must have valid port and TCP flag information, specify **--reject-nonzero-fragments --reject-incomplete**.

--set-sensorid=*SCALAR*

Set the sensor ID for all flows to *SCALAR*. *SCALAR* should be an integer value between 0 and 65534, inclusive. When not specified, the sensor ID is set to 65535.

--set-inputindex=*SCALAR*

Set the input SNMP index value for all flows to *SCALAR*. *SCALAR* should be an integer value between 0 and 65535, inclusive. When not specified, the SNMP input is set to 0.

--set-outputindex=*SCALAR*

Set the output SNMP index value for all flows to *SCALAR*. *SCALAR* should be an integer value between 0 and 65535, inclusive. When not specified, the SNMP output is set to 0.

--set-nexthopip=*IP_ADDRESS*

Set the next-hop IP address for all flows to *IP_ADDRESS*; *IP_ADDRESS* may be in its canonical form or an integer. When not specified, the next-hop IP is set to 0.0.0.0.

--print-statistics

Print a summary of the packets that were processed. This summary includes

- the total number of packets read
- the number that fell outside the time-window
- the number that were too short to get the IP header
- the number that were not IPv4
- the number that were discarded by a plug-in
- the total number of fragmented packets
- the number of fragments where the offset was zero
- the number of zero-offset packets that were incomplete
- the number of flows written to the output

--note-add=*TEXT*

Add the specified *TEXT* to the header of the output file as an annotation. This switch may be repeated to add multiple annotations to a file. To view the annotations, use the **rwfileinfo(1)** tool.

--note-file-add=*FILENAME*

Open *FILENAME* and add the contents of that file to the header of the output file as an annotation. This switch may be repeated to add multiple annotations. Currently the application makes no effort to ensure that *FILENAME* contains text; be careful that you do not attempt to add a SiLK data file as an annotation.

--compression-method=*COMP_METHOD*

Specify the compression library to use when writing output files. If this switch is not given, the value in the SILK_COMPRESSION_METHOD environment variable is used if the value names an available compression method. When no compression method is specified, output to the standard output or to named pipes is not compressed, and output to files is compressed using the default chosen when SiLK was compiled. The valid values for *COMP_METHOD* are determined by which external libraries were found when SiLK was compiled. To see the available compression methods and the default method, use the **--help** or **--version** switch. SiLK can support the following *COMP_METHOD* values when the required libraries are available.

none

Do not compress the output using an external library.

zlib

Use the **zlib(3)** library for compressing the output, and always compress the output regardless of the destination. Using zlib produces the smallest output files at the cost of speed.

lzo1x

Use the *lzo1x* algorithm from the LZO real time compression library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead.

snappy

Use the *snappy* library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead. *Since SiLK 3.13.0.*

best

Use lzo1x if available, otherwise use snappy if available, otherwise use zlib if available. Only compress the output when writing to a file.

--help

Print the available options and exit. Options that add fields can be specified before **--help** so that the new options appear in the output.

--version

Print the version number and information about how SiLK was configured, then exit the application.

PLUG-IN SUPPORT

rwptoflow allows the user to provide additional logic to ignore or reject packets, or to modify the flow record that is generated from the packet. To do this, the user creates a **plug-in** that gets loaded at run-time by giving **rwptoflow** the **--plugin** switch with the path to the plug-in as the parameter to the switch.

A plug-in is a shared object file (a.k.a. dynamic library) that is compiled from C source code. The plug-in should have four subroutines defined:

setup()

is called when the object is first loaded. This is the place to initialize global variables to their default values. If the plug-in provides switches of its own, they must be registered in this subroutine.

initialize()

gets called after all options have been processed but before any packets are read from the input. If this subroutine does not return 0, the application will quit.

ptoflow()

will be called for every packet that **rwptoflow** is able to convert into a flow record just before the flow record is written. This subroutine will *not* see packets that are short or that are not IPv4; it will also not see fragmented packets if **--reject-all-fragments** is specified.

The `ptoflow()` function is called with two parameters:

- a pointer to the `rwRec` object that **rwptoflow** created from the packet. The subroutine may modify the record as it sees fit.
- a void pointer that the function may cast to a pointer to the C structure:

```
typedef struct _sk_pktsrc_t {
    /* the source of the packets */
    pcap_t                      *pcap_src;
    /* the pcap header as returned from pcap_next() */
    const struct pcap_pkthdr    *pcap_hdr;
    /* the packet as returned from pcap_next() */
    const u_char                *pcap_data;
} sk_pktsrc_t;
```

This structure gives the user access to all the information about the packet.

The following return values from `ptoflow()` determines whether **rwptoflow** writes the flow and the packet:

0

Write the flow record to the **flow-output** and the packet to the *PCKTS_PASS* unless another plug-in instructs otherwise.

1

Write the flow record to the **flow-output** and the packet to the *PCKTS_PASS* immediately; do not call the `ptoflow()` routine on any other plug-in.

2

Treat the packet as a reject: Do *not* write the flow record; write the packet to the *PCKTS_REJECT* immediately; do not call the `ptoflow()` routine on any other plug-in.

3

Ignore the packet immediately: Do *not* write the flow record nor the packet; do not call the `ptoflow()` routine on any other plug-in.

If `ptoflow()` returns any other value, the **rwptoflow** application will terminate with an error.

teardown()

is called as the application exits. The user can use this routine to print results and to free() any data structures that were used.

rwptoflow uses the following rules to find the plug-in: When *PLUGIN* contains a slash (`/`), **rwptoflow** assumes the path to *PLUGIN* is correct. Otherwise, **rwptoflow** will attempt to find the file in *$SILK_PATH/lib/silk*, *$SILK_PATH/share/lib*, *$SILK_PATH/lib*, and in these directories parallel to the application's directory: *lib/silk*, *share/lib*, and *lib*. If **rwptoflow** does not find the file, it assumes the plug-in is in the current directory. To force **rwptoflow** to look in the current directory first, specify **--plugin**=./*PLUGIN*. When the SILK_PLUGIN_DEBUG environment variable is non-empty, **rwptoflow** prints status messages to the standard error as it tries to open each of its plug-ins.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line.

Given the packet capture file *data.pcap*, convert it to a SiLK flow file, *data.rw*, and copy the packets that **rwptoflow** understands to the file *good.pcap*:

```
$ rwptoflow data.pcap --packet-pass=good.pcap --flow-out=data.rw
```

Use **rwfilter(1)** to partition the SiLK Flows records, writing those records whose source IPs are found in the IPset file *sip.set* to *filtered.rw*:

```
$ rwfilter --sipset=sip.set --pass=filtered.rw  data.rw
```

Use **rwpmatch(1)** to match the capture file, *good.pcap*, against the filtered SiLK file, in affect generating a capture file which has been filtered by *sip.set*:

```
$ rwpmatch --flow-file=filtered.rw good.pcap > filtered.pcap
```

ENVIRONMENT

SILK_PLUGIN_DEBUG

When set to 1, **rwptoflow** print status messages to the standard error as it tries to open each of its plug-ins.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_COMPRESSION_METHOD

This environment variable is used as the value for **--compression-method** when that switch is not provided. *Since SiLK 3.13.0.*

SILK_PATH

This environment variable gives the root of the install tree. When searching for a plug-in, **rwptoflow** may use this environment variable.

SEE ALSO

rwpmatch(1), **rwpdedupe(1)**, **rwfilter(1)**, **rwfileinfo(1)**, **rwp2yaf2silk(1)**, **rwipfix2silk(1)**, **silk(7)**, **yaf(1)**, **tcpdump(1)**, **pcap(3)**, **mergecap(1)**, **zlib(3)**

NOTES

SiLK supports millisecond timestamps. When reading packets whose timestamps have finer precision, the times are truncated at the millisecond position.

The **mergecap(1)** or **rwpdedupe(1)** programs can be used to join multiple **tcpdump** capture files in order to convert into a single flow file.

rwrandomizeip

Randomize the IP addresses in a SiLK Flow file

SYNOPSIS

```
rwrandomizeip [--seed=NUMBER] [--only-change-set=CHANGE_IPSET]
      [--dont-change-set=KEEP_IPSET]
      [--consistent] [--save-table=FILE] [--load-table=FILE]
      [--site-config-file=FILENAME]
      [INPUT_FILE [OUTPUT_FILE]]

rwrandomizeip --help

rwrandomizeip --version
```

DESCRIPTION

Read SiLK Flow records from *INPUT_FILE*, substitute a pseudo-random IP address for the source and destination IP addresses, and write the result to *OUTPUT_FILE*.

rwrandomizeip reads its input from the standard input either when no non-switch arguments are given or when *INPUT_FILE* is the string `stdin` or `-`. **rwrandomizeip** writes its output to the standard output either when the number of non-switch arguments is less than two or when *OUTPUT_FILE* is the string `stdout` or `-`. Since **rwrandomizeip** processes binary data, it exits with an error if either *INPUT_FILE* or *OUTPUT_FILE* refer to a terminal. **rwrandomizeip** is able to read and write files that have been compressed with **gzip(1)** when the file name ends with `.gz`.

To only change a subset of the IP addresses, the optional switches **--only-change-set** or **--dont-change-set** may be used; each switch takes an IPset file as its required argument. When the **--only-change-set**=*CHANGE_IPSET* switch is given, **rwrandomizeip** modifies only the IP addresses listed in the *CHANGE_IPSET* file. To change all addresses *except* a specified set, use **rwsetbuild(1)** to create an IPset file containing those IPs and pass the name of the file to the **--dont-change-set** switch. An address listed in both the **only-change-set** and the **dont-change-set** is not modified. When the same IPset is passed to the **--only-change-set** and **--dont-change-set** switches, the output is identical to the input for all records.

The **--seed** switch may be used to initialize the pseudo-random number generator to a known state.

Use of the **--consistent**, **--load-table**, or **--save-table** switches causes **rwrandomizeip** to operate in consistent mode. When none of the switches are specified, it operates in inconsistent mode.

Consistent Mode

In consistent mode, the octets of an IPv4 address are modified such that structural information of the data is maintained, and multiple instances of an input IP address are mapped to the same randomized output address. Unfortunately, this comes at a cost of less randomness in the output. Specifically, **rwrandomizeip** creates four internal tables with each table having 256 entries containing the values 0--255 that have been randomly shuffled. When an IP address is read, each table is used to map the values for a specific octet of that IP address. For example, when modifying the IP address 10.10.10.10, the value at position 10 from each table is substituted into the IP.

Inconsistent Mode

In this mode, **rwrandomizeip** uses a pseudo-random address for each source and destination IP address it processes. Each record is handled individually, and an IP address that appears multiple times in the input file is mapped to a different output address each time. Thus, no structural information in the input is maintained. **rwrandomizeip** changes each IP address to a non-routable address from the CIDR blocks 10.0.0.0/8, 172.16.0.0/12, or 192.168.0.0/16.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--seed=*NUMBER*

Use *NUMBER* to seed the pseudo-random number generator. This may be used to put the random number generator into a known state, which is useful for testing.

--only-change-set=*CHANGE_IPSET*

Only modify the source or destination IP address if it appears in the given IPset file *CHANGE_IPSET*. The **rwsetbuild** command may be used to create an IPset file. When the **--dont-change-set**=*KEEP_IPSET* switch is also given, the IPs it contains override those in the *CHANGE_IPSET* file.

--dont-change-set=*KEEP_IPSET*

Do not modify the source or destination IP address if the address appears in the given IPset file *KEEP_IPSET*. The **rwsetbuild** command may be used to create an IPset file. The interaction of this switch with the **--only-change-set** switch is described immediately above.

--consistent

Randomize the IP addresses consistently, so that an input IP address is always mapped to the same value. The default behavior is to use a random IP address for each IP, even if the IP has been seen before.

--save-table=*FILE*

Randomize the IP addresses consistently and save this run's randomization table for future use. The table is written to the specified *FILE*, which must not not exist. This switch is incompatible with the **--load-table** switch.

--load-table=*FILE*

Randomize the IP addresses consistently using the randomization table contained in *FILE* that was created by a previous invocation of **rwrandomizeip**. This switch is incompatible with the **--save-table** switch.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwrandomizeip** searches for the site configuration file in the locations specified in the FILES section.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

ENVIRONMENT

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This variable gives the root of the directory tree where the data store of SiLK Flow files is maintained, overriding the location that is compiled into the tools (/data). **rwswapbytes** may search for the site configuration file, *silk.conf*, in this directory. See the FILES section for details.

SILK_PATH

This environment variable gives the root of the directory tree where the tools are installed. As part of its search for the site configuration file, **rwrandomizeip** may use this variable. See the FILES section for details.

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file.

SEE ALSO

rwsetbuild(1), **silk(7)**

BUGS

rwrandomizeip does not support IPv6 flow records. When an input file contains IPv6 records, **rwrandomizeip** converts records that contain addresses in the ::ffff:0:0/96 prefix to IPv4 and processes them. **rwrandomizeip** silently ignores IPv6 records containing addresses outside of that prefix.

Only the source and destination IP fields are modified; additional fields in the SiLK Flow records may leak sensitive information.

Prior to SiLK 3.16.0, **rwrandomizeip** required explicit arguments for the input file and the output file.

rwrecgenerator

Generate random SiLK Flow records

SYNOPSIS

```
rwrecgenerator { --silk-output-path=PATH | --text-output-path=PATH
                 | { --output-directory=DIR_PATH
                     --processing-directory=DIR_PATH }}
      --log-destination=DESTINATION [--log-level=LEVEL]
      [--log-sysfacility=NUMBER] [--seed=SEED]
      [--start-time=START_DATETIME --end-time=END_DATETIME]
      [--time-step=MILLISECONDS] [--events-per-step=COUNT]
      [--num-subprocesses=COUNT] [--flush-timeout=MILLISEC]
      [--file-cache-size=SIZE] [--compression-method=COMP_METHOD]
      [--timestamp-format=FORMAT] [--epoch-time]
      [--ip-format=FORMAT] [--integer-ips] [--zero-pad-ips]
      [--integer-sensors] [--integer-tcp-flags] [--no-titles]
      [--no-columns] [--column-separator=CHAR]
      [--no-final-delimiter] [--delimited=[CHAR]]]
      [--site-config-file=FILENAME] [--sensor-prefix-map=FILE]
      [--flowtype-in=CLASS/TYPE] [--flowtype-inweb=CLASS/TYPE]
      [--flowtype-out=CLASS/TYPE] [--flowtype-outweb=CLASS/TYPE]

rwrecgenerator --help

rwrecgenerator --version
```

DESCRIPTION

rwrecgenerator uses pseudo-random numbers to generate *events*, where each consists of one or more SiLK Flow records. These flow records can written as a single binary file, as text (in either a columnar or a comma separated value format) similar to the output from **rwcut(1)**, or as a directory of small binary files to mimic the *incremental files* produced by **rwflowpack(8)**. The type of output to produce must be specified using the appropriate switches. Currently only one type of output may be produced in a single invocation.

rwrecgenerator works through a time window, where the starting and ending times for the window may be specified on the command line. When not specified, the window defaults to the previous hour. By default, **rwrecgenerator** will generate one event at the start time and one event at the end time. To modify the size of the steps **rwrecgenerator** takes across the window, specify the **--time-step** switch. The number of events to create at each step may be specified with the **--events-per-step** switch.

The time window specifies when the events begin. Since most events create multiple flow records with small time offsets between them (and some events may create flow records across multiple hours), flow records will exist that begin after the time window.

To generate a single SiLK flow file, specify its location with the **--silk-output-path** switch. A value of `-` will write the output to the standard output unless the standard output is connected to a terminal.

To produce textual output, specify **--text-output-path**. **rwrecgenerator** has numerous switches to control the appearance of the text; however, currently **rwrecgenerator** produces a fixed set of fields.

When creating incremental files, the **--output-directory** and **--processing-directory** switches are required. **rwrecgenerator** creates files in the processing directory, and moves the files to the output directory when the flush timeout arrives. The default flush timeout is 30,000 milliseconds (30 seconds); the user may modify the value with the **--flush-timeout** switch. Any files in the processing directory are removed when **rwrecgenerator** starts.

The **--num-subprocesses** switch tells **rwrecgenerator** to use multiple subprocesses when creating incremental files. When the switch is specified, **rwrecgenerator** will split the time window into multiple pieces and give each subprocess its own time window to create. The initial **rwrecgenerator** process then waits for the subprocesses to complete. When **--num-subprocesses** is specified, **rwrecgenerator** will create subdirectories under the **--processing-directory**, where each subprocess gets its own processing directory.

The **--seed** switch may be specified to provide a consistent set of flow records across multiple invocations. (Note that the names of the incremental files will differ across invocations since those names are created with the mkstemp(3) function.)

Given the same seed for the pseudo-random number generator and assuming the **--num-subprocesses** is *not* specified, the output from **rwrecgenerator** will contain the same data regardless of whether the output is written to a single SiLK flow file, a text file, or a series of incremental files.

When both **--seed** and **--num-subprocesses** is specified, the incremental files will contain the same flow records across invocations, but the flow records will not be consistent with those created by **--silk-output-path** or **--text-output-path**.

rwrecgenerator must have access to a **silk.conf(5)** site configuration file, either specified by the **--site-config-file** switch on the command line or specified by the typical methods.

The **--flowtype-in**, **--flowtype-inweb**, **--flowtype-out**, and **--flowtype-outweb** switches may be used to specify the *flowtype* (that is, the *class/type* pair) that **rwrecgenerator** uses for its flow records. When these switches are not specified, **rwrecgenerator** attempts to use the flowtypes defined in the *silk.conf* file for the *twoway* site. Specifically, it attempts to use "all/in", "all/inweb", "all/out", and "all/outweb", respectively.

Use of the **--sensor-prefix-map** switch is recommended. The argument should name a prefix map file that maps from an internal IP address to a sensor number. If the switch is not provided, all flow records will use the first sensor in the *silk.conf* file that is supported by the class specified by the flowtypes. When using the **--sensor-prefix-map**, make certain the sensors you choose are in the class specified in the **--flowtype-**** switches.

When using the **--sensor-prefix-map** switch and creating incremental files, it is recommended that you use the **--file-cache-size** switch to increase the size of the stream cache to be approximately 12 to 16 times the number of sensors. This will reduce the amount of time spent closing and reopening the files.

The **--log-destination** switch is required. Specify **none** to disable logging.

Currently, **rwrecgenerator** only supports generating IPv4 addresses. Addresses in 0.0.0.0/1 are considered internal, and addresses in 128.0.0.0/1 are considered external. All flow records are between an internal and an external address. Whether the internal addresses is the source or destination of the unidirectional flow record is determined randomly.

The types of flow records that **rwrecgenerator** creates are:

- HTTP traffic on port 80/tcp that consists of a query and a response. This traffic will be about 30% of the total by flow count.
- HTTPS traffic on port 443/tcp that consists of a query and a response. This traffic will be about 30% of the total by flow count.

- DNS traffic on port 53/udp that consists of a query and a response. This traffic will be about 10% of the total by flow count.
- FTP traffic on port 21/tcp that consists of a query and a response. This traffic will be about 4% of the total by flow count.
- ICMP traffic on that consists of a single message. This traffic will be about 4% of the total by flow count.
- IMAP traffic on port 143/tcp that consists of a query and a response. This traffic will be about 4% of the total by flow count.
- POP3 traffic on port 110/tcp that consists of a query and a response. This traffic will be about 4% of the total by flow count.
- SMTP traffic on port 25/tcp that consists of a query and a response. This traffic will be about 4% of the total by flow count.
- TELNET traffic on port 23/tcp between two machines. This traffic may involve multiple flow records that reach the active timeout of 1800 seconds. This traffic will be about 4% of the total by flow count.
- Traffic on IP Protocols 47, 50, or 58 that consists of a single record. This traffic will be about 4% of the total by flow count.
- Scans of every port on one IP address. This traffic will be about 1% of the total by flow count.
- Scans of a single port across a range of IP addresses. This traffic will be about 1% of the total by flow count.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

Output Switches

Exactly one of the following switches is required.

--silk-output-path=*PATH*

Tell **rwrecgenerator** to create a single binary file of SiLK flow records at the specified location. If *PATH* is **-**, the records are written to the standard output. **rwrecgenerator** does not support writing binary data to a terminal.

--output-directory=*DIR_PATH*

Name the directory into which the incremental files are written once the flush timeout is reached.

--text-output-path=*PATH*

Tell **rwrecgenerator** to convert the flow records it creates to text and to print the result in a format similar to that created by **rwcut(1)**. The output is written to the specified location. If *PATH* is **-**, the records are written to the standard output.

Logging Switches

The **--log-destination** switch is required. Use a value of **none** to disable logging.

--log-destination=*DESTINATION*

Specify the destination where logging messages are written. When *DESTINATION* begins with a slash `/`, it is treated as a file system path and all log messages are written to that file; there is no log rotation. When *DESTINATION* does not begin with `/`, it must be one of the following strings:

`none`
: Messages are not written anywhere.

`stdout`
: Messages are written to the standard output.

`stderr`
: Messages are written to the standard error.

`syslog`
: Messages are written using the **syslog(3)** facility.

`both`
: Messages are written to the syslog facility and to the standard error (this option is not available on all platforms).

--log-level=*LEVEL*

Set the severity of messages that will be logged. The levels from most severe to least are: `emerg`, `alert`, `crit`, `err`, `warning`, `notice`, `info`, `debug`. The default is `info`.

--log-sysfacility=*NUMBER*

Set the facility that **syslog(3)** uses for logging messages. This switch takes a number as an argument. The default is a value that corresponds to `LOG_USER` on the system where **rwrecgenerator** is running. This switch produces an error unless **--log-destination**=syslog is specified.

General Switches

The following are general purpose switches. None are required.

--seed=*SEED*

Seed the pseudo-random number generator with the value *SEED*. When not specified, **rwrecgenerator** creates its own seed. Specifying the seed allows different invocations of **rwrecgenerator** to produce the same output (assuming the same value is given for all switches and that the time window is specified).

--start-time=*YYYY/MM/DD[:HH[:MM[:SS[.ssssss]]]]*

--start-time=*EPOCH_SECONDS_PLUS_MILLISECONDS*

Specify the earliest date and time at which an event is started. The specified time must be given to at least day precision. Any parts of the date-time string that are not specified are set to 0. The switch also accepts UNIX epoch seconds with optional fractional seconds. When not specified, defaults to the beginning of the previous hour.

--end-time=*YYYY/MM/DD[:HH[:MM[:SS[.ssssss]]]]*

--end-time=*EPOCH_SECONDS_PLUS_MILLISECONDS*

Specify the latest date and time at which an event is started. This time does **not** specify the latest end-time for the flow records or even the latest start-time, since many events simulate a query/response pair, with the response following the query by a few milliseconds. The specified time must be given to at least day precision, and it must not be less than the start-time. Any parts of the date-time string that are not specified are set to 0. The switch also accepts UNIX epoch seconds with optional fractional seconds. When not specified, defaults to the end of the previous hour.

--time-step=*MILLISECONDS*

Move forward *MILLISECONDS* milliseconds at each step as **rwrecgenerator** moves through the time window. When not specified, defaults to the difference between the start-time and end-time; that is, **rwrecgenerator** will generate events at the start-time and then at the end-time. A *MILLISECONDS* value of 0 indicates **rwrecgenerator** should only create events at the start-time.

--events-per-step=*COUNT*

Create *COUNT* events at each time step. The default is 1.

--help

Print the available options and exit.

--version

Print the version number and information about how **rwrecgenerator** was configured, then exit the application.

Incremental Files Switches

The following switches are used when creating incremental files.

--processing-directory=*DIR_PATH*

Name the directory under the incremental files are initially created. Any files in this directory are removed when **rwrecgenerator** is started. When the flush timeout is reached, the files are closed and moved from this directory to the output-directory. If **--num-subprocesses** is specified, subdirectories are created under *DIR_PATH*, and each subprocess is given its own subdirectory.

--num-subprocesses=*COUNT*

Tell **rwrecgenerator** to create *COUNT* subprocesses to generate incremental files. This switch is ignored when incremental files are not being created. When this switch is specified, **rwrecgenerator** creates subdirectories below the processing directory. The default value for *COUNT* is 0.

--flush-timeout=*MILLISECONDS*

Set the timeout for flushing any in-memory records to disk to *MILLISECONDS* milliseconds. At this time, the incremental files are closed and the files are moved from the processing directory to the output directory. The timeout uses the internal time as **rwrecgenerator** moves through the time window. If not specified, the default is 30,000 milliseconds (30 seconds). This switch is ignored when incremental files are not being created.

--file-cache-size=*SIZE*

Set the maximum number of data files to have open for writing at any one time to *SIZE*. If not specified, the default is 32 files.

--compression-method=*COMP_METHOD*

Specify the compression library to use when writing binary output files. If this switch is not given, the value in the SILK_COMPRESSION_METHOD environment variable is used if the value names an available compression method. When no compression method is specified, binary output is compressed using the default chosen when SiLK was compiled. The valid values for *COMP_METHOD* are determined by which external libraries were found when SiLK was compiled. To see the available compression methods and the default method, use the **--help** or **--version** switch. SiLK can support the following *COMP_METHOD* values when the required libraries are available.

none
Do not compress the SiLK Flow records using an external library.

zlib
Use the **zlib(3)** library for compressing the flow records.

lzo1x
Use the *lzo1x* algorithm from the LZO real-time compression library for compressing the flow records.

snappy
Use the *snappy* library for compressing the flow records. *Since SiLK 3.13.0.*

best
Use lzo1x if available, otherwise use snappy if available, otherwise use zlib if available.

Text File Switches

The following switches can be used when creating textual output.

--timestamp-format=*FORMAT*

When producing textual output, specify the format, timezone, and/or modifier to use when printing timestamps. When this switch is not specified, the SILK_TIMESTAMP_FORMAT environment variable is checked for a format, timezone, and modifier. If it is empty or contains invalid values, timestamps are printed in the default format, and the timezone is UTC unless SiLK was compiled with local timezone support. *FORMAT* is a comma-separated list of a format, a timezone, and/or a modifier. The format is one of:

default
Print the timestamps as *YYYY/MM/DD*T*hh*:*mm*:*ss.sss*.

iso
Print the timestamps as *YYYY-MM-DD hh*:*mm*:*ss.sss*.

m/d/y
Print the timestamps as *MM/DD/YYYY hh*:*mm*:*ss.sss*.

epoch
Print the timestamps as the number of seconds since 00:00:00 UTC on 1970-01-01.

When a timezone is specified, it is used regardless of the default timezone support compiled into SiLK. The timezone is one of:

utc
Use Coordinated Universal Time to print timestamps.

local
Use the TZ environment variable or the local timezone.

One modifier is available:

no-msec
Truncate the milliseconds value on the timestamps and on the duration field. When milliseconds are truncated, the sum of the printed start time and duration may not equal the printed end time.

--epoch-time

When producing textual output, print timestamps as epoch time (number of seconds since midnight GMT on 1970-01-01). This switch is equivalent to **--timestamp-format=epoch**, it is deprecated as of SiLK 3.8.1, and it will be removed in the SiLK 4.0 release.

--ip-format=*FORMAT*

When producing textual output, specify how IP addresses are printed, where *FORMAT* is a comma-separated list of the arguments described below. When this switch is not specified, the SILK_IP_FORMAT environment variable is checked for a value and that format is used if it is valid. The default *FORMAT* is `canonical`. *Since SiLK 3.8.1.*

canonical
Print IP addresses in the canonical format. For an IPv4 record, use dot-separated decimal (`192.0.2.1`). For an IPv6 records, use either colon-separated hexadecimal (`2001:db8::1`) a or mixed IPv4-IPv6 representation for IPv4-mapped IPv6 addresses (the ::ffff:0:0/96 netblock, e.g., `::ffff:192.0.2.1`) and IPv4-compatible IPv6 addresses (the ::/96 netblock other than ::/127, e.g., `::192.0.2.1`).

no-mixed
Print IP addresses in the canonical format (`192.0.2.1` or `2001:db8::1`) but do not used the mixed IPv4-IPv6 representations. For example, use `::ffff:c000:201` instead of `::ffff:192.0.2.1`. *Since SiLK 3.17.0.*

decimal
Print IP addresses as integers in decimal format. For example, print `192.0.2.1` and `2001:db8::1` as `3221225985` and `42540766411282592856903984951653826561`, respectively.

hexadecimal
Print IP addresses as integers in hexadecimal format. For example, print `192.0.2.1` and `2001:db8::1` as `c00000201` and `20010db8000000000000000000000001`, respectively.

zero-padded
Make all IP address strings contain the same number of characters by padding numbers with leading zeros. For example, print `192.0.2.1` and `2001:db8::1` as `192.000.002.001` and `2001:0db8:0000:0000:0000:0000:0000:0001`, respectively. For IPv6 addresses, this setting implies `no-mixed`, so that `::ffff:192.0.2.1` is printed as `0000:0000:0000:0000:0000:ffff:c000:0201`. As of SiLK 3.17.0, may be combined with any of the above, including `decimal` and `hexadecimal`.

The following arguments modify certain IP addresses prior to printing. These arguments may be combined with the above formats.

map-v4
Change IPv4 addresses to IPv4-mapped IPv6 addresses (addresses in the ::ffff:0:0/96 netblock) prior to formatting. *Since SiLK 3.17.0.*

unmap-v6

Change any IPv4-mapped IPv6 addresses (addresses in the ::ffff:0:0/96 netblock) to IPv4 addresses prior to formatting. *Since SiLK 3.17.0.*

The following argument is also available:

force-ipv6

Set *FORMAT* to `map-v4`,`no-mixed`.

--integer-ips

When producing textual output, print IP addresses as integers. This switch is equivalent to **--ip-format=decimal**, it is deprecated as of SiLK 3.8.1, and it will be removed in the SiLK 4.0 release.

--zero-pad-ips

When producing textual output, print IP addresses as fully-expanded, zero-padded values in their canonical form. This switch is equivalent to **--ip-format=zero-padded**, it is deprecated as of SiLK 3.8.1, and it will be removed in the SiLK 4.0 release.

--integer-sensors

When producing textual output, print the integer ID of the sensor rather than its name.

--integer-tcp-flags

When producing textual output, print the TCP flag fields (flags, initialFlags, sessionFlags) as an integer value. Typically, the characters `F,S,R,P,A,U,E,C` are used to represent the TCP flags.

--no-titles

When producing textual output, turn off column titles. By default, titles are printed.

--no-columns

When producing textual output, disable fixed-width columnar output.

--column-separator=*C*

When producing textual output, use specified character between columns and after the final column. When this switch is not specified, the default of '|' is used.

--no-final-delimiter

When producing textual output, do not print the column separator after the final column. Normally a delimiter is printed.

--delimited

--delimited=*C*

When producing textual output, run as if **--no-columns --no-final-delimiter --column-sep**=*C* had been specified. That is, disable fixed-width columnar output; if character *C* is provided, it is used as the delimiter between columns instead of the default '|'.

SiLK Site Specific Switches

The following switches control the class/type and sensor that **rwrecgenerator** assigns to every flow record.

--sensor-prefix-map=*FILE*

Load a prefix map from *FILE* and use it to map from the internal IP addresses to sensor numbers. If the switch is not provided, all flow records will use the first sensor in the *silk.conf* file that is supported by the class named in the flowtype. The sensor IDs specified in *FILE* should agree with the class specified in the **--flowtype-***** switches.

--flowtype-in=*CLASS/TYPE*

Set the class/type pair for flow records where the source IP is external, the destination IP is internal, and the flow record is not considered to represent a *web record* to *CLASS/TYPE*. Web records are those that appear on ports 80/tcp, 443/tcp, and 8080/tcp. When not specified, **rwrecgenerator** attempts to find the flowtype "all/in" in the *silk.conf* file.

--flowtype-inweb=*CLASS/TYPE*

Set the class/type pair for flow records representing web records where the source IP is external and the destination IP is internal to *CLASS/TYPE*. When not specified and the **--flowtype-in** switch is given, that *CLASS/TYPE* pair will be used. When neither this switch nor **--flowtype-in** is given, **rwrecgenerator** attempts to find the flowtype "all/inweb" in the *silk.conf* file.

--flowtype-out=*CLASS/TYPE*

Set the class/type pair for flow records where the source IP is internal, the destination IP is external, and the flow record is not considered to represent a *web record* to *CLASS/TYPE*. When not specified, **rwrecgenerator** attempts to find the flowtype "all/out" in the *silk.conf* file.

--flowtype-outweb=*CLASS/TYPE*

Set the class/type pair for flow records representing web records where the source IP is internal and the destination IP is external to *CLASS/TYPE*. When not specified and the **--flowtype-out** switch is given, that *CLASS/TYPE* pair will be used. When neither this switch nor **--flowtype-out** is given, **rwrecgenerator** attempts to find the flowtype "all/outweb" in the *silk.conf* file.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwrecgenerator** searches for the site configuration file in the locations specified in the FILES section.

ENVIRONMENT

SILK_IP_FORMAT

This environment variable is used as the value for **--ip-format** when that switch is not provided. *Since SiLK 3.11.0.*

SILK_TIMESTAMP_FORMAT

This environment variable is used as the value for **--timestamp-format** when that switch is not provided. *Since SiLK 3.11.0.*

SILK_COMPRESSION_METHOD

This environment variable is used as the value for **--compression-method** when that switch is not provided. *Since SiLK 3.13.0.*

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwrecgenerator** may use this environment variable when searching for the SiLK site configuration file.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **rwrecgenerator** may use this environment variable. See the FILES section for details.

TZ

When the argument to the **--timestamp-format** switch includes `local` or when a SiLK installation is built to use the local timezone, the value of the TZ environment variable determines the timezone in which **rwrecgenerator** displays timestamps. (If both of those are false, the TZ environment variable is ignored.) If the TZ environment variable is not set, the machine's default timezone is used. Setting TZ to the empty string or 0 causes timestamps to be displayed in UTC. For system information on the TZ variable, see **tzset(3)** or **environ(7)**. (To determine if SiLK was built with support for the local timezone, check the `Timezone support` value in the output of **rwrecgenerator --version**.) The TZ environment variable is also used when **rwrecgenerator** parses the timestamp specified in the **--start-time** or **--end-time** switches if SiLK is built with local timezone support.

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

SEE ALSO

silk(7), **rwcut(1)**, **rwflowpack(8)**, **silk.conf(5)**, **syslog(3)**, **zlib(3)**, **tzset(3)**, **environ(7)**

rwresolve

Convert IP addresses in delimited text to hostnames

SYNOPSIS

```
rwresolve [--ip-fields=FIELDS] [--delimiter=C] [--column-width=N]
    [--resolver={ c-ares | adns | getnameinfo | gethostbyaddr }]
    [--max-requests=N]

rwresolve --help

rwresolve --version
```

DESCRIPTION

rwresolve is an application that reads delimited textual input and maps IP addresses in the input to host names up performing a reverse DNS look-up. If the look-up succeeds, the IP is replaced with the host name (**rwresolve** uses the first host name returned by the resolver). If the look-up fails, the IP address remains unchanged.

rwresolve does a DNS query for every IP address, so it can be **extremely** slow. **rwresolve** works best on very limited data sets. To reduce the number of DNS calls it makes, **rwresolve** caches the results of queries. There are two libraries that support asynchronous DNS queries which **rwresolve** can use if either of those libraries was found when SiLK was configured. These libraries are the ADNS library and the c-ares library. Specify the **--resolver** switch to have **rwresolve** use a particular function for look-ups.

When an IP address resolves to multiple names, **rwresolve** prints the first name returned by the resolver.

rwresolve is designed specifically to deal with the output of **rwcut(1)**, though it will work with other SiLK tools that produce delimited text. **rwresolve** reads the standard input, splits the line into fields based on the delimiter (default '|'), converts the specified *FIELDS* (default fields 1 and 2) from an IP address in its canonical form (e.g., dotted decimal for IPv4) to a hostname. If the field cannot be parsed as an address or if the look up fails to return a hostname, the field is not modified. The fields to convert are specified via the **--ip-fields**=*FIELDS* option. The **--delimiter** option can be used to specify an alternate delimiter.

Since hostnames are generally wider than IP addresses, the use of the **--column-width** field is advised to increase the width of the IP columns. If this switch is not specified, no justification of hostnames is attempted.

By default, **rwresolve** will use the c-ares library if available, then it will use the ADNS library if available. To choose a different IP look up option, use the **--resolver** switch.

The maximum number of parallel DNS queries to attempt with c-ares or ADNS can be specified with the **--max-requests** switch.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--ip-fields=*FIELDS*

Specify the column number(s) of the input that should be considered IP addresses. Column numbers start from 1. If not specified, the default is 1,2.

--delimiter=*C*

Specify the character that separates the columns of the input. The default is '|'.

--column-width=*WIDTH*

Set the width of the columns specified in **--ip-fields** to *WIDTH*. When specified, the *FIELDS* columns always have the specified *WIDTH* regardless of whether the IP to hostname mapping was successful. If this switch is not specified, fields containing IP addresses that could not be resolved will maintain their input length, and fields where the lookup was successful will be printed with no padding.

--resolver=c-ares

Use the c-ares library to convert the IP addresses to hostnames. Requires that the c-ares library was found when SiLK was configured. This library supports IPv6 look-ups when SiLK is compiled to support IPv6.

--resolver=adns

Use the ADNS library to convert the IP addresses to hostnames. Requires that the ADNS library was found when SiLK was configured. This library only supports IPv4 look-ups.

--resolver=getnameinfo

Use the **getnameinfo(3)** C library function to convert IP addresses to hostnames. This function supports IPv6 look-ups when SiLK is compiled to support IPv6.

--resolver=gethostbyaddr

Use the **gethostbyaddr(3)** C library function to convert IP addresses to hostnames. This function only supports IPv4.

--max-requests=*MAX*

When the c-ares or ADNS library is used, limit the number of outstanding DNS queries active at any one time to *MAX*. The default is 128. This switch is not available if neither c-ares nor ADNS were found when SiLK was compiled.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLE

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

Suppose you have found some interesting data in the file *interesting.rw*, and you want to view the data using **rwcut(1)**, but you also want to determine the hostname of each the source IPs and append that hostname to the **rwcut** output. In the example command below, note how the source IP field (**rwcut** field 1) was specified twice in the **rwcut** invocation, and **rwresolve** is told to resolve the second occurrence, which is field in column 13. This allows you to see the source IP (in the first column) and the host name it mapped to (in the final column).

```
$ rwcut --fields=1-12,1 interesting.rw             \
  | rwresolve --ip-field=13
```

ENVIRONMENT

When ADNS is used, the following environment variables affect it. The ADNS_ form of each variable takes precedence.

RES_CONF

ADNS_RES_CONF

A filename, whose contents are in the format of *resolv.conf*.

RES_CONF_TEXT

ADNS_RES_CONF_TEXT

A string in the format of *resolv.conf*.

RES_OPTIONS

ADNS_RES_OPTIONS

These are parsed as if they appeared in the **options** line of a *resolv.conf*. In addition to being parsed at this point in the sequence, they are also parsed at the very beginning before *resolv.conf* or any other environment variables are read, so that any debug option can affect the processing of the configuration.

LOCALDOMAIN

ADNS_LOCALDOMAIN

These are interpreted as if their contents appeared in a **search** line in *resolv.conf*.

SEE ALSO

rwcut(1), **silk(7)**, **gethostbyaddr(3)**, **getnameinfo(3)**

BUGS

Because **rwresolve** must do a DNS query for every IP address, it is **extremely** slow.

The output from **rwresolve** is rarely columnar because hostnames can be very long. You may want to consider putting the resolved hostnames in the final column of output.

rwscan

Detect scanning activity in a SiLK dataset

SYNOPSIS

```
rwscan [--scan-model=MODEL] [--output-path=PATH]
       [--trw-internal-set=SETFILE]
       [--trw-theta0=PROB] [--trw-theta1=PROB]
       [--no-titles] [--no-columns] [--column-separator=CHAR]
       [--no-final-delimiter] [{--delimited | --delimited=CHAR}]
       [--integer-ips] [--model-fields] [--scandb]
       [--threads=THREADS] [--queue-depth=DEPTH]
       [--verbose-progress=CIDR] [--verbose-flows]
       [ {--verbose-results | --verbose-results=NUM} ]
       [--site-config-file=FILENAME]
       [FILES...]

rwscan --help

rwscan --version
```

DESCRIPTION

rwscan reads sorted SiLK Flow records, performs scan detection analysis on those records, and outputs textual columnar output for the scanning IP addresses. **rwscan** writes its out to the **--output-path** or to the standard output when **--output-path** is not specified.

The types of scan detection analysis that **rwscan** supports are Threshold Random Walk (TRW) and Bayesian Logistic Regression (BLR). Details about these techniques are described in the METHOD OF OPERATION section below.

rwscan is designed to write its data into a database. This database can be queried using the **rwscan-query(1)** tool. See the EXAMPLES section for the recommended database schema.

The input to **rwscan** should be pre-sorted using **rwsort(1)** by the source IP, protocol, and destination IP (i.e., **--fields=sip,proto,dip**).

rwscan reads SiLK Flow records from the files named on the command line or from the standard input when no file names are specified. To read the standard input in addition to the named files, use `-` or `stdin` as a file name. If an input file name ends in `.gz`, the file is uncompressed as it is read.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--scan-model=*MODEL*

Select a specific scan detection model. If not specified, the default value for *MODEL* is `0`. See the METHOD OF OPERATION section for more details.

0

Use the Threshold Random Walk (TRW) and Bayesian Logistic Regression (BLR) scan detection models in series.

1

Use only the TRW scan detection model.

2

Use only the BLR scan detection model.

--output-path=*PATH*

Write the textual output to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output (and bypass the paging program). If *PATH* names an existing file, **rwscan** exits with an error unless the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. If this switch is not given, the output is either sent to the pager or written to the standard output.

--trw-internal-set=*SETFILE*

Specify an IPset file containing **all** valid internal IP addresses. This parameter is required when using the TRW scan detection model, since the TRW model requires the list of targeted IPs (i.e., the IPs to detect the scanning activity to). This switch is ignored when the TRW model is not used. For information on creating IPset files, see the **rwset(1)** and **rwsetbuild(1)** manual pages. Prior to SiLK 3.4, this switch was named **--trw-sip-set**.

--trw-sip-set=*SETFILE*

This is a deprecated alias for **--trw-internal-set**.

--trw-theta0=*PROB*

Set the theta_0 parameter for the TRW scan model to *PROB*, which must be a floating point number between 0 and 1. theta_0 is defined as the probability that a connection succeeds given the hypothesis that the remote source is benign (not a scanner). The default value for this option is 0.8. This option should only be used by experts familiar with the TRW algorithm.

--trw-theta1=*PROB*

Set the theta_1 parameter for the TRW scan model to *PROB*, which must be a floating point number between 0 and 1. theta_1 is defined as the probability that a connection succeeds given the hypothesis that the remote source is malicious (a scanner). The default value for this option is 0.2. This option should only be used by experts familiar with the TRW algorithm.

--no-titles

Turn off column titles. By default, titles are printed.

--no-columns

Disable fixed-width columnar output.

--column-separator=*C*

Use specified character between columns. When this switch is not specified, the default of '|' is used.

--no-final-delimiter

Do not print the column separator after the final column. Normally a delimiter is printed.

--delimited

--delimited=*C*

Run as if **--no-columns --no-final-delimiter --column-sep**=*C* had been specified. That is, disable fixed-width column output; if character *C* is provided, it is used as the delimiter between columns instead of the default '|'.

--integer-ips

Print IP addresses as decimal integers instead of in their canonical representation.

--model-fields

Show scan model detail fields. This switch controls whether additional informational fields about the scan detection models are printed.

--scandb

Produce output suitable for loading into a database. Sample database schema are given below under EXAMPLES. This option is equivalent to **--no-titles --no-columns --no-final-delimiter --model-fields --integer-ips**.

--threads=*THREADS*

Specify the number of worker threads to create for scan detection processing. By default, one thread will be used. Changing this number to match the number of available CPUs will often yield a large performance improvement.

--queue-depth=*DEPTH*

Specify the depth of the work queue. The default is to make the work queue the same size as the number of worker threads, but this can be changed. Normally, the default is fine.

--verbose-progress=*CIDR*

Report progress as **rwscan** processes input data. The *CIDR* argument should be an integer that corresponds to the netblock size of each line of progress. For example, **--verbose-progress**=*8* would print a progress message for each /8 network processed.

--verbose-flows

Cause **rwscan** to print very verbose information for each flow. This switch is primarily useful for debugging.

--verbose-results

--verbose-results=*NUM*

Print detailed information on each IP processed by **rwscan**. If a *NUM* argument is provided, only print verbose results for sources that sent at least *NUM* flows. This information includes scan model calculations, overall scan scores, etc. This option will generate a lot of output, and is primarily useful for debugging.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwscan** searches for the site configuration file in the locations specified in the FILES section.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

METHOD OF OPERATION

rwscan's default behavior is to consult two scan detection models to determine whether a source is a scanner. The primary model used is the Threshold Random Walk (TRW) model. The TRW algorithm takes advantage of the tendency of scanners to attempt to contact a large number of IPs that do not exist on the target network.

By keeping track of the number of "hits" (successful connections) and "misses" (attempts to connect to IP addresses that are not active on the target network), scanners can be detected quickly and with a high degree of accuracy. Sequential hypothesis testing is used to analyze the probability that a source is a scanner as each flow record is processed. Once the scan probability exceeds a configured maximum, the source is flagged as a scanner, and no further analysis of traffic from that host is necessary.

The TRW model is not 100% accurate, however, and only finds scans in TCP flow data. In the case where the TRW model is inconclusive, a secondary model called BLR is invoked. BLR stands for "Bayesian Logistic Regression." Unlike TRW, the BLR approach must analyze all traffic from a given source IP to determine whether that IP is a scanner.

Because of this, BLR operates much slower than TRW. However, the BLR model has been shown to detect scans that are not detected by the TRW model, particularly scans in UDP and ICMP data, and vertical TCP scans which focus on finding services on a single host. It does this by calculating metrics from the flow data from each source, and using those metrics to arrive at an overall likelihood that the flow data represents scanning activity.

The metrics BLR uses for detecting scans in TCP flow data are:

- the ratio of flows with no ACK bit set to all flows
- the ratio of flows with fewer than three packets to all flows
- the average number of source ports per destination IP address
- the ratio of the number of flows that have an average of 60 bytes/packet or greater to all flows
- the ratio of the number of unique destination IP addresses to the total number of flows
- the ratio of the number of flows where the flag combination indicates backscatter to all flows

The metrics BLR uses for detecting scans in UDP flow data are:

- the ratio of flows with fewer than three packets to all flows
- the maximum run length of IP addresses per /24 subnet
- the maximum number of unique low-numbered (less than 1024) destination ports contacted on any one host
- the maximum number of consecutive low-numbered destination ports contacted on any one host
- the average number of unique source ports per destination IP address
- the ratio of flows with 60 or more bytes/packet to all flows
- the ratio of unique source ports (both low and high) to the number of flows

The metrics BLR uses for detecting scans in ICMP flow data are:

- the maximum number of consecutive /24 subnets that were contacted
- the maximum run length of IP addresses per /24 subnet
- the maximum number of IP addresses contacted in any one /24 subnet
- the total number of IP addresses contacted
- the ratio of ICMP echo requests to all ICMP flows

Because the TRW model has a lower false positive rate than the BLR model, any source identified as a scanner by TRW will be identified as a scanner by the hybrid model without consulting BLR. BLR is only invoked in the following cases:

- The traffic being analyzed is UDP or ICMP traffic, which **rwscan**'s implementation of TRW cannot process.
- The TRW model has identified the source as benign. This occurs when the scan probability drops below a configured minimum during sequential hypothesis testing.
- The TRW model has identified the source as unknown (where the scan probability never exceeded the minimum or maximum thresholds during sequential hypothesis testing).

In situations where the use of one model is preferred, the other model can be disabled using the **--scan-model** switch. This may have an impact on the performance and/or accuracy of the system.

LIMITATIONS

rwscan detects scans in IPv4 flows only.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

Basic Usage

Assuming a properly sorted SiLK Flow file as input, the basic usage for Bayesian Logistic Regression (BLR) scan detection requires only the input file, *data.rw*, and output file, *scans.txt*, arguments.

```
$ rwscan --scan-model=2 --output-path=scans.txt data.rw
```

Basic usage of Threshold Random Walk (TRW) scan detection requires the IP addresses of the targeted network (i.e., the internal IP space), specified in the *internal.set* IPset file.

```
$ rwscan --trw-internal-set=internal.set --output-path=scans.txt data.rw
```

Typical Usage

More commonly, an analyst uses **rwfilter(1)** to query the data repository for flow records within a time window. First, the analyst has **rwset(1)** put the source addresses of *outgoing* flow records into an IPset, resulting in the IPset containing the IPs of active hosts on the internal network. Next, the *incoming* traffic is piped to **rwsort(1)** and then to **rwscan**.

```
$ rwfilter --start=2004/12/29:00 --type=out,outweb --all-dest=stdout \
  | rwset --sip=internal.set

$ rwfilter --start=2004/12/29:00 --type=in,inweb --all-dest=stdout \
  | rwsort --fields=sip,proto,dip                                   \
  | rwscan --trw-internal-set=internal.set --scan-model=0           \
       --output-path=scans.txt
```

Storing Scans in a PostgreSQL Database

Instead of having the analyst run **rwscan** directly, often the output from **rwscan** is put into a database where it can be queried by **rwscanquery(1)**. The output produced by the **--scandb** switch is suitable for loading into a database of scans. The process for using the PostgreSQL database is described in this section.

Schemas for Oracle, MySQL, and SQLite are provided below, but the details to create users with the proper rolls are not included.

Here is the schema for PostgreSQL:

```
CREATE DATABASE scans

CREATE SCHEMA scans

CREATE SEQUENCE scans_id_seq

CREATE TABLE scans (
  id          BIGINT      NOT NULL    DEFAULT nextval('scans_id_seq'),
  sip         BIGINT      NOT NULL,
  proto       SMALLINT    NOT NULL,
  stime       TIMESTAMP without time zone NOT NULL,
  etime       TIMESTAMP without time zone NOT NULL,
  flows       BIGINT      NOT NULL,
  packets     BIGINT      NOT NULL,
  bytes       BIGINT      NOT NULL,
  scan_model  INTEGER     NOT NULL,
  scan_prob   FLOAT       NOT NULL,
  PRIMARY KEY (id)
)

CREATE INDEX scans_stime_idx ON scans (stime)
CREATE INDEX scans_etime_idx ON scans (etime)
;
```

A database user should be created for the purposes of populating the scan database, e.g.:

```
CREATE USER rwscan WITH PASSWORD 'secret';

GRANT ALL PRIVILEGES ON DATABASE scans TO rwscan;
```

Additionally, a user with read-only access should be created for use by the **rwscanquery** tool:

```
CREATE USER rwscanquery WITH PASSWORD 'secret';

GRANT SELECT ON DATABASE scans TO rwscanquery;
```

To import **rwscan**'s **--scandb** output into a PostgreSQL database, use a command similar to the following:

```
$ cat /tmp/scans.import.txt            \
  | psql -c                            \
    "COPY scans                        \
        (sip, proto, stime, etime,     \
        flows, packets, bytes,         \
        scan_model, scan_prob)         \
    FROM stdin DELIMITER as '|'" scans
```

Sample Schema for Oracle

```
CREATE TABLE scans (
  id          integer unsigned    not null unique,
  sip         integer unsigned    not null,
  proto       tinyint unsigned    not null,
  stime       datetime            not null,
  etime       datetime            not null,
  flows       integer unsigned    not null,
  packets     integer unsigned    not null,
  bytes       integer unsigned    not null,
  scan_model  integer unsigned    not null,
  scan_prob   float unsigned      not null,
  primary key (id)
);
```

Sample Schema for MySQL

```
CREATE TABLE scans (
  id          integer unsigned    not null auto_increment,
  sip         integer unsigned    not null,
  proto       tinyint unsigned    not null,
  stime       datetime            not null,
  etime       datetime            not null,
  flows       integer unsigned    not null,
  packets     integer unsigned    not null,
  bytes       integer unsigned    not null,
  scan_model  integer unsigned    not null,
  scan_prob   float unsigned      not null,
```

```
  primary key (id),
  INDEX (stime),
  INDEX (etime)
) TYPE=InnoDB;
```

Sample Schema and Import Command for SQLite

```
CREATE TABLE scans (
  id          INTEGER PRIMARY KEY AUTOINCREMENT,
  sip         INTEGER             NOT NULL,
  proto       SMALLINT            NOT NULL,
  stime       TIMESTAMP           NOT NULL,
  etime       TIMESTAMP           NOT NULL,
  flows       INTEGER             NOT NULL,
  packets     INTEGER             NOT NULL,
  bytes       INTEGER             NOT NULL,
  scan_model  INTEGER             NOT NULL,
  scan_prob   FLOAT               NOT NULL
);
CREATE INDEX scans_stime_idx ON scans (stime);
CREATE INDEX scans_etime_idx ON scans (etime);
```

To import **rwscan**'s **--scandb** output into a SQLite database, use the following command:

```
$ perl -nwe 'chomp;
    print "INSERT INTO scans VALUES (NULL,",
          (join ",",map { / / ? qq("$_") : $_ } split /\|/),
          ");\n";' \
scans.txt | sqlite3 scans.sqlite
```

ENVIRONMENT

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwscan** may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **rwscan** may use this environment variable. See the FILES section for details.

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

SEE ALSO

rwscanquery(1), **rwfilter(1)**, **rwsort(1)**, **rwset(1)**, **rwsetbuild(1)**, **silk(7)**

BUGS

When used in an IPv6 environment, **rwscan** converts IPv6 flow records that contain addresses in the ::ffff:0:0/96 prefix to IPv4. IPv6 records outside of that prefix are silently ignored.

rwscanquery

Query the network scan database

SYNOPSIS

```
rwscanquery [options]
```

Report Options:

```
  --report=REPORT_TYPE       Select query and output options.  Values
                             for REPORT_TYPE are standard, volume,
                             scanset, scanflows, respflows, and export

  --start-date=YYYY/MM/DD:HH Report on scans active after this date.
  --end-date=YYYY/MM/DD:HH   Defaults to start-date.

  --saddress=ADDR_SPEC       Show scans originating from matching hosts.
  --sipset=IPSET_FILE        Show scans originating from hosts in set.

  --daddress=IP_WILDCARD     Show only scans targeting matching hosts.
  --dipset=IPSET_FILE        Show only scans targeting hosts in set.

  --show-header              Display column titles at start of output.
  --columnar                 Display more human-readable columnar view.
  --output-path=PATH         Write results to the specified file.
```

Configuration Options:

```
  --database=DBNAME          Query an alternate scan database
```

Help Options:

```
  --help                     Display this brief help message.
  --man                      Display the full documentation.
  --version                  Display the version information.
```

DESCRIPTION

rwscanquery queries the network scan database---that is, the database that contains scans found by **rwscan(1)**. The type of output **rwscanquery** creates is controlled by the **--report** switch as described in the Report Options section below. **rwscanquery** writes its output to the location specified by the **--output-path** switch or to the standard output when that switch is not provided.

rwscanquery runs a query of the scan database and then, depending on the report type, either displays the result set as text or creates a binary SiLK from the result set. The database rows that are part of the result set may be limited by using the **--start-date**, **--end-date**, **--saddress**, and **--sipset** switches. The result set is always limited to a time window, and the current day is used when no **--start-date** is given.

The following three report types produce textual output. The default output displays the values separated by a vertical bar (|) with no spacing. The **--columnar** switch causes the output to appear in columns with a space-delimiter between the columns. The output includes no title line unless the **--show-header** switch is specified.

- The `standard` report contains most of the columns in the database for the rows in the result set. (The columns containing the scan model and scan probability are not included.)
- The `volume` report groups the rows in the result set by day and shows sums the flows, packets, and bytes columns for each day.
- The `export` report contains all the columns in the database for the rows in the result set, and the rows are displayed in a format compatible with **rwscan**.

The following three report types create a binary SiLK file as their result. These report types invoke other SiLK tools (namely **rwfilter(1)**, **rwset(1)**, **rwsetbuild(1)**, and **rwsetcat(1)**) and the report types assume **rwfilter** has access to a SiLK data repository.

The first step in all three of these report types is for **rwscanquery** to get the distinct IP addresses for the rows in the result set and pass them into **rwsetbuild** to create a temporary IPset file containing the scanning IPs.

- A `scanflows` report produces a file of SiLK Flow records whose source IP is a scanning IPs. **rwscanquery** uses the temporary IPset as an argument to **rwfilter** to find flow records in your data repository that originated from the scanning IPs within the time window. You may choose to limit the report to particular IPs targeted by the scanning IPs by specifying the **--daddress** or **--dipset** switches. The output from **rwfilter** is the output of the report. The **rwfilter** invocation uses the configuration values `rw_in_class` and `rw_in_type` if they are specified in the configuration file (c.f. CONFIGURATION).
- A `respflows` report produces a file of SiLK Flow records whose destination IP is a scanning IP. These flow records may represent responses to a scan. To create this report, **rwscanquery** performs steps similar to those for the `scanflows` report except the direction of the **rwfilter** command is reversed to find flow records going to the scanning IPs. You may choose to limit the report to particular IPs that responded to the scan by specifying the **--daddress** or **--dipset** switches. The output from **rwfilter** is the output of the report. The **rwfilter** invocation uses the configuration values `rw_out_class` and `rw_out_type` if they are specified in the configuration file (c.f. CONFIGURATION).
- A `scanset` report produces a binary IPset file.
  - If neither the **--daddress** nor **--dipset** switches are specified, the output of the this report is the temporary IPset file containing the scanning IPs; that is, all the scanning IPs in the time window.
  - Otherwise, **rwscanquery** performs the same steps it does as when creating `scanflows` report. Next, instead of returning the output from **rwfilter**, **rwscanquery** passes the flow records into **rwset** to create an IPset file containing the scanning IPs that targeted particular IP addresses.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

Report Options

--report=*TYPE*

Specify the query and the type of output to create. When this switch is not specified, the default is a `standard` report. The supported values for *TYPE* are:

standard

Write one textual line of output for each scan record in the scan database. By default, the output has no titles and it is not in columnar form. Specify the **--show-header** and/or **--columnar** switches to make the output more human readable.

volume

Write a daily scan activity volume summary report for each day within the time period. By default, the output has no titles and it is not in columnar form. Specify the **--show-header** and/or **--columnar** switches to make the output more human readable.

scanset

Write an IPset file containing the IP addresses which were the sources of scan activity during the selected time period. The output of this report type is binary, so you must redirect or pipe the output to a location or specify the **--output-path** switch.

scanflows

Write a SiLK Flow file containing all flows originating from scanning IP addresses within the specified time period. This flow data includes flows originating from any host that would be listed as a scan source by your query, from any time within the time period specified by *--start-date* and *--end-date*. Note that this may include flows that were not identified by the scan analysis as being part of a scan. The output of this report type is binary, so you must redirect or pipe the output to a location or specify the **--output-path** switch.

respflows

Write a SiLK Flow file containing all flows sent to scanning IP addresses within the specified time period---that is, possible responses to the scanners. The output of this report type is binary, so you must redirect or pipe the output to a location or specify the **--output-path** switch.

export

Write textual output consistent with the output format of the **rwscan(1)** tool. Specify the **--show-header** switch to include a title line.

--start-date=*YYYY/MM/DD:HH*

Display scans which were active after this hour. When this argument contains a date with no hour and no **--end-date** switch is specified, scans for that entire day are returned. If this switch is not specified at all, scans for the current day (based on the local time on the host machine) are returned.

--end-date=*YYYY/MM/DD:HH*

Display scans which were active before the end of this hour. If no end-date is given, defaults to the same as start-date. It is an error to provide an end-date without a start-date.

--saddress=*ADDR_SPEC*

Display scans originating from hosts described in *ADDR_SPEC*, where *ADDR_SPEC* is a list of addresses, address ranges, and CIDR blocks. Only scans originating from hosts in the list are displayed.

--sipset=*IPSET_FILE*

Display scans originating from hosts in *IPSET_FILE*, where *IPSET_FILE* is a standard SiLK IPset file as created by **rwset(1)** or **rwsetbuild(1)**. Note that a very complex IPset may take a long time to process, or even fail to return any results.

--daddress=*IP_WILDCARD*

Display scans targeting hosts described in *IP_WILDCARD*, where *IP_WILDCARD* is a single IP address, a single CIDR block, or an IP Wildcard expression accepted by **rwfilter(1)**. To match on multiple IPs or networks, use the **--dipset** switch. This switch is ignored for **--report** types other than `scanset`, `scanflows`, and `respflows`.

--dipset=*IPSET_FILE*

Display scans targeting hosts in *IPSET_FILE*, where *IPSET_FILE* is a standard SiLK IPset file. This switch is ignored for **--report** types other than `scanset`, `scanflows`, and `respflows`.

--show-header

Display a header line giving a short name (or title) for each field when printing textual output with the `standard`, `volume`, or `export` report types. By default, no header is displayed.

--columnar

Display output in more human-readable columnar format when printing textual output with the `standard` or `volume` report types. When this switch is not given, the output is presented as data fields delimited by the | character.

--output-path=*PATH*

Write results to *PATH* instead of to the standard output.

Configuration Options

--database=*DBNAME*

Select a database instance other than the default. The default is specified by the `db_instance` value in the configuration file as described in CONFIGURATION below.

Other Options

--help

Display a brief usage message and exit.

--man

Display full documentation for **rwscanquery** and exit.

--version

Print the version number and exit the application.

CONFIGURATION

rwscanquery reads configuration information from a file named *.rwscanrc*. If the RWSCANRC environment variable is set, it is used as the location of the *.rwscanrc* file. When RWSCANRC is not set, **rwscanquery** attempts to find a file name *.rwscanrc* in the directories specified in the FILES section below.

The format of the *.rwscanrc* file is *name=value* pairs, one per line. The configuration parameters currently read from *.rwscanrc* are:

db_driver

The type of database to connect to. **rwscanquery** supports `oracle`, `postgresql`, `mysql`, and `sqlite`.

db_userid

The userid to use when connecting to the scan database.

db_password

The password to use when connecting to the scan database.

db_instance

The name of the database instance to connect to if none is provided with the **--database** command line switch. If neither this configuration option nor the **--database** command line switch are specified, the hard-coded default database instance "SCAN" is used.

rw_in_class

The class for incoming flow data. The `rw_in_class` and `rw_in_type` values are used to query scan flows when the `scanflows` report type is requested or when the **--daddress** or **--dipset** switches are used for the `scanset` report type. If not specified, **rwfilter**'s default is used.

rw_in_type

The type(s) for incoming flow data. See `rw_in_class` for details.

rw_out_class

The class for outgoing flow data. The `rw_out_class` and `rw_out_type` values are used to query scan flows when the `respflows` report type is requested. If not specified, **rwfilter**'s default is used.

rw_out_type

The type(s) for outgoing flow data. See `rw_out_class` for details. (Note that **rwfilter** often defaults to querying incoming flows, so this parameter ought to be specified.)

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

Display information on all scans occurring during the 12:00 hour (12:00:00 to 12:59:59) of 2009/02/12.

```
$ rwscanquery --show-header --start-date=2009/02/12:12
scan-id|stime|etime|proto|srcaddr|flows|packets|bytes
499|2009-02-12 12:01:56|2009-02-12 12:08:39|6|10.199.151.231|256|256|10240
365|2009-02-12 12:08:40|2009-02-12 12:14:54|6|10.146.88.117|256|256|10240
57|2009-02-12 12:28:51|2009-02-12 12:34:55|6|10.29.23.160|256|256|10240
627|2009-02-12 11:52:07|2009-02-12 12:41:16|17|10.253.24.230|1023|1023|30175
366|2009-02-12 12:41:50|2009-02-12 12:48:14|6|10.146.89.46|256|256|10240
182|2009-02-12 12:54:39|2009-02-12 13:01:20|6|10.79.26.176|256|256|10240
4|2009-02-12 12:41:19|2009-02-12 13:33:57|17|10.2.47.87|1023|1023|30205
```

Create the IPset file *scan.set* containing the scanners discovered during that hour.

```
$ rwscanquery --report=scanset --start-date=2009/02/12:12 \
       --output-path=scan.set
$ rwsetcat scan.set
10.2.47.87
```

```
10.29.23.160
10.79.26.176
10.146.88.117
10.146.89.46
10.199.151.231
10.253.24.230
```

Repeat the first query but limit the output to scanners coming from the CIDR block 10.199.0.0/16.

```
$ rwscanquery --show-header --start-date=2009/02/12:12 \
       --saddr=10.199.0.0/16
scan-id|stime|etime|proto|srcaddr|flows|packets|bytes
499|2009-02-12 12:01:56|2009-02-12 12:08:39|6|10.199.151.231|256|256|10240
```

Expand the query for that CIDR block to include the preceding and following hours (11:00:00 to 13:59:59).

```
$ rwscanquery --start-date=2009/02/12:11 --end-date=2009/02/12:13 \
       --saddr=10.199.0.0/16
499|2009-02-12 12:01:56|2009-02-12 12:08:39|6|10.199.151.231|256|256|10240
497|2009-02-12 13:33:57|2009-02-12 14:24:35|17|10.199.98.5|1023|1023|30079
```

Create the IPset file *scanning-cidr.set* that contains the CIDR block 10.199.0.0/16, and then search for scans coming from that IP on Feb 13, 2009.

```
$ cat scanning-cidr.txt
10.199.0.0/16
$ rwsetbuild scanning-cidr.txt scanning-cidr.set
$
$ rwscanquery --start-date=2009/02/13 --sipset=scanning-cidr.set
500|2009-02-13 22:42:25|2009-02-13 22:48:45|6|10.199.207.32|256|256|10240
```

Print the volume of data attributed to scans over a three day period.

```
$ rwscanquery --report=volume --show-header  \
       --start-date=2009/02/12 --end-date=2009/02/14
date|flows|packets|bytes
2009/02/12|137452|137499|17149008
2009/02/13|74727|76167|2798040
2009/02/14|76160|76160|2750531
```

The following limits the volume report to the IPs in the file *scanning-cidr.set* and displays the results in columns.

```
$ rwscanquery --report=volume --show-header --columnar  \
       --start-date=2009/02/12 --end-date=2009/02/14    \
       --sipset=scanning-cidr.set
date                    flows    packets          bytes
2009/02/12               1279       1279          40319
2009/02/13                256        256          10240
2009/02/14                256        256          10240
```

Get the SiLK Flow records coming from the scanners during the 12:00 hour on 2009/02/12 and store in the file *scanning-flows.rw*.

```
$ rwscanquery --report=scanflows --start-date=2009/02/12:12  \
       --output=scanning-flows.rw
```

Use **rwuniq(1)** to summarize the file *scanning-flows.rw*.

```
$ rwuniq --fields=sip --values=flows,packets,bytes  \
       --sort-output scanning-flows.rw
            sIP|   Records|        Packets|               Bytes|
     10.2.47.87|       373|            373|               11032|
   10.29.23.160|       256|            256|               10240|
   10.79.26.176|       203|            203|                8120|
  10.146.88.117|       256|            256|               10240|
   10.146.89.46|       256|            256|               10240|
  10.199.151.231|       256|            256|               10240|
   10.253.24.230|       846|            846|               24921|
```

Run a respflows report to verify that there were no responses to the scan.

```
$ rwscanquery --report=respflows --start-date=2009/02/12:12  \
       --output=scanning-response.rw
$
$ rwuniq --fields=sip --values=flows,packets,bytes  \
       --sort-output scanning-response.rw
            sIP|   Records|        Packets|               Bytes|
```

Create the IPset *subnet-scan.set* for scanners that targeted the 192.168.186.0/24 CIDR block during the 12:00 hour on 2009/02/12.

```
$ rwscanquery --report=scanset --start-date=2009/02/12:12        \
       --daddress=192.168.186.0/24 --output-path=subnet-scan.set
```

Store the corresponding flow records for those scans in the file *subset-scan.rw*.

```
$ rwscanquery --report=scanflows --start-date=2009/02/12:12        \
       --daddress=192.168.186.0/24 --output-path=subnet-scan.rw
```

Determine how many IPs in that subnet were targeted.

```
$ rwuniq --fields=sip --values=flows,distinct:dip subnet-scan.rw
            sIP|   Records|dIP-Distin|
   10.146.89.46|       256|       256|
```

Display the title line for an export report.

```
$ rwscanquery --report=export --start-date=2009/02/12:12  \
       --show-header | head -1
id|sip|proto|stime|etime|flows|packets|bytes|scan_model|scan_prob
```

ENVIRONMENT

RWSCANRC

This environment variable allows the user to specify the location of the *.rwscanrc* configuration file. The value may be a complete path or a file relative to the user's current directory. See the FILES section for standard locations of this file.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction for the report types of `scanset`, `scanflows`, and `respflows`.

SILK_CONFIG_FILE

This environment variable is used as the location for the site configuration file, *silk.conf*, for report types that use **rwfilter**. When this environment variable is not set, **rwfilter** searches for the site configuration file in the locations specified in the FILES section.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository for report types that use **rwfilter**. This value overrides the compiled-in value. In addition, **rwfilter** may use this value when searching for the SiLK site configuration files. See the FILES section for details.

SILK_RWFILTER_THREADS

The number of threads **rwfilter** uses when reading files from the data store.

SILK_PATH

This environment variable gives the root of the install tree. When searching for the site configuration file, **rwfilter** may use this environment variable. See the FILES section for details.

PATH

This is the standard UNIX path (c.f., **environ(7)**). Depending on the report type, **rwscanquery** may invoke **rwfilter(1)**, **rwset(1)**, **rwsetbuild(1)**, or **rwsetcat(1)** as part of its processing.

RWFILTER

Complete path to **rwfilter**. If not set, **rwscanquery** attempts to find **rwfilter** on your PATH.

RWSET

Complete path to **rwset**. If not set, **rwscanquery** attempts to find **rwset** on your PATH.

RWSETBUILD

Complete path to **rwsetbuild**. If not set, **rwscanquery** attempts to find **rwsetbuild** on your PATH.

RWSETCAT

Complete path to **rwsetcat**. If not set, **rwscanquery** attempts to find **rwsetcat** on your PATH.

FILES

${RWSCANRC}

${HOME}/.rwscanrc

/usr/local/share/silk/.rwscanrc

Possible locations for the **rwscanquery** configuration file, *.rwscanrc*. In addition, **rwscanquery** checks the parent directory of the directory containing the **rwscanquery** script.

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file---for report types that use **rwfilter**.

SEE ALSO

rwscan(1), **rwfilter(1)**, **rwset(1)**, **rwsetbuild(1)**, **rwsetcat(1)**, **rwuniq(1)**, **silk(7)**, **environ(7)**

rwset

Generate binary IPset files of unique IP addresses

SYNOPSIS

```
rwset {--sip-file=FILE | --dip-file=FILE
       | --nhip-file=FILE | --any-file=FILE [...]}
      [--record-version=VERSION] [--invocation-strip]
      [--note-strip] [--note-add=TEXT] [--note-file-add=FILE]
      [--print-filenames] [--copy-input=PATH]
      [--compression-method=COMP_METHOD]
      [--ipv6-policy={ignore,asv4,mix,force,only}]
      [--site-config-file=FILENAME]
      {[--xargs] | [--xargs=FILENAME] | [FILE [FILE ...]]}

rwset --help

rwset --version
```

DESCRIPTION

rwset reads SiLK Flow records and generates one to four binary IPset file(s). In a single pass, **rwset** can create one of each type of its possible outputs, which are IPset files containing:

- the unique source IP addresses
- the unique destination IP addresses
- the unique next-hop IP addresses
- the unique source and destination IP addresses

The output files must not exist prior to invoking **rwset**. To write an IPset file to the standard output, specify `stdout` or `-` as the output file name. **rwset** will complain if you attempt to write the IPset to the standard output and standard output is connected to the terminal. Only one IPset file may be written to the standard output.

rwset reads SiLK Flow records from the files named on the command line or from the standard input when no file names are specified and **--xargs** is not present. To read the standard input in addition to the named files, use `-` or `stdin` as a file name. If an input file name ends in `.gz`, the file is uncompressed as it is read. When the **--xargs** switch is provided, **rwset** reads the names of the files to process from the named text file or from the standard input if no file name argument is provided to the switch. The input to **--xargs** must contain one file name per line.

IPset files are in a binary format that efficiently stores a set of IP addresses. The file only stores the presence of an IP address; no volume information (such as a count of the number of times the IP address occurs) is maintained. To store volume information, use **rwbag(1)**.

Use **rwsetcat(1)** to see the IP addresses in a binary IPset file. To create a binary IPset file from a list of IP addresses, use **rwsetbuild(1)**. **rwsettool(1)** allows you to perform set operations on binary IPset files. To determine if an IP address is a member of a binary IPset, use **rwsetmember(1)**.

To list the IPs that appear in the SiLK Flow file *flows.rw*, the command

```
$ rwset --sip-file=stdout flows.rw | rwsetcat
```

is faster than **rwuniq(1)**, but **rwset** does not report the number of flow records or compute byte and packets counts.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

At least one of the following output switches is required; multiple output switches can be given, but an output switch cannot be repeated.

--sip-file=*FILE*

Store the unique source IP addresses in the binary IPset file *FILE*. **rwset** will write the IPset file to the standard output when *FILE* is `stdout` or `-` and the standard output is not a terminal.

--dip-file=*FILE*

Store the unique destination IP addresses in the binary IPset file *FILE*. **rwset** will write the IPset file to the standard output when *FILE* is `stdout` or `-` and the standard output is not a terminal.

--nhip-file=*FILE*

Store the unique next-hop IP addresses in the binary IPset file *FILE*. **rwset** will write the IPset file to the standard output when *FILE* is `stdout` and the standard output is not a terminal.

--any-file=*FILE*

Store the unique source and destination IP addresses in the binary IPset file *FILE*. **rwset** will write the IPset file to the standard output when *FILE* is `stdout` or `-` and the standard output is not a terminal.

Only one of the above switches my use `stdout` as the name of the file.

rwset supports these additional switches:

--record-version=*VERSION*

Specify the format of the IPset records that are written to the output. *VERSION* may be 2, 3, 4, 5 or the special value 0. When the switch is not provided, the SILK_IPSET_RECORD_VERSION environment variable is checked for a version. The default version is 0.

0

Use the default version for an IPv4 IPset and an IPv6 IPset. Use the **--help** switch to see the versions used for your SiLK installation.

2

Create a file that may hold only IPv4 addresses and is readable by all versions of SiLK.

3

Create a file that may hold IPv4 or IPv6 addresses and is readable by SiLK 3.0 and later.

4

Create a file that may hold IPv4 or IPv6 addresses and is readable by SiLK 3.7 and later. These files are more compact that version 3 and often more compact than version 2.

5

Create a file that may hold only IPv6 addresses and is readable by SiLK 3.14 and later. When this version is specified, IPsets containing only IPv4 addresses are written in version 4. These files are usually more compact that version 4.

--invocation-strip

Do not record any command line history: do not copy the invocation history from the input files to the output file, and do not record the current command line invocation in the output. The invocation may be viewed with **rwfileinfo(1)**.

--note-strip

Do not copy the notes (annotations) from the input files to the output file. Normally notes from the input files are copied to the output.

--note-add=*TEXT*

Add the specified *TEXT* to the header of every output file as an annotation. This switch may be repeated to add multiple annotations to a file. To view the annotations, use the **rwfileinfo(1)** tool.

--note-file-add=*FILENAME*

Open *FILENAME* and add the contents of that file to the header of every output file as an annotation. This switch may be repeated to add multiple annotations. Currently the application makes no effort to ensure that *FILENAME* contains text; be careful that you do not attempt to add a SiLK data file as an annotation.

--print-filenames

Print to the standard error the names of input files as they are opened.

--copy-input=*PATH*

Copy all binary SiLK Flow records read as input to the specified file or named pipe. *PATH* may be `stdout` or `-` to write flows to the standard output as long as no IPset file is being written there.

--ipv6-policy=*POLICY*

Determine how IPv4 and IPv6 flows are handled when SiLK has been compiled with IPv6 support. When the switch is not provided, the SILK_IPV6_POLICY environment variable is checked for a policy. If it is also unset or contains an invalid policy, the *POLICY* is **mix**. When SiLK has not been compiled with IPv6 support, IPv6 flows are always ignored, regardless of the value passed to this switch or in the SILK_IPV6_POLICY variable. The supported values for *POLICY* are:

ignore

Ignore any flow record marked as IPv6, regardless of the IP addresses it contains. Only IP addresses contained in IPv4 flow records will be added to the IPset(s).

asv4

Convert IPv6 flow records that contain addresses in the ::ffff:0:0/96 netblock (that is, IPv4-mapped IPv6 addresses) to IPv4 and ignore all other IPv6 flow records.

mix

Process the input as a mixture of IPv4 and IPv6 flow records. When the input contains IPv6 addresses outside of the ::ffff:0:0/96 netblock, this policy is equivalent to **force**; otherwise it is equivalent to **asv4**.

force
Convert IPv4 flow records to IPv6, mapping the IPv4 addresses into the ::ffff:0:0/96 netblock.

only
Process only flow records that are marked as IPv6. Only IP addresses contained in IPv6 flow records will be added to the IPset(s).

Regardless of the IPv6 policy, when all IPv6 addresses in the IPset are in the ::ffff:0:0/96 netblock, **rwset** treats them as IPv4 addresses and writes an IPv4 IPset. When any other IPv6 addresses are present in the IPset, the IPv4 addresses in the IPset are mapped into the ::ffff:0:0/96 netblock and **rwset** writes an IPv6 IPset.

--compression-method=*COMP_METHOD*

Specify the compression library to use when writing output files. If this switch is not given, the value in the SILK_COMPRESSION_METHOD environment variable is used if the value names an available compression method. When no compression method is specified, output to the standard output or to named pipes is not compressed, and output to files is compressed using the default chosen when SiLK was compiled. The valid values for *COMP_METHOD* are determined by which external libraries were found when SiLK was compiled. To see the available compression methods and the default method, use the **--help** or **--version** switch. SiLK can support the following *COMP_METHOD* values when the required libraries are available.

none
Do not compress the output using an external library.

zlib
Use the **zlib(3)** library for compressing the output, and always compress the output regardless of the destination. Using zlib produces the smallest output files at the cost of speed.

lzo1x
Use the *lzo1x* algorithm from the LZO real time compression library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead.

snappy
Use the *snappy* library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead. *Since SiLK 3.13.0.*

best
Use lzo1x if available, otherwise use snappy if available, otherwise use zlib if available. Only compress the output when writing to a file.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwset** searches for the site configuration file in the locations specified in the FILES section.

--xargs

--xargs=*FILENAME*

Read the names of the input files from *FILENAME* or from the standard input if *FILENAME* is not provided. The input is expected to have one filename per line. **rwset** opens each named file in turn and reads records from it as if the filenames had been listed on the command line.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

rwset is intended to work tightly with **rwfilter(1)**. For example, consider generating two IPsets: the first file, *low_packet_tcp.set*, contains the source IP addresses for incoming flow records (that is, the external hosts) where the record has no more than three packets in its sessions. The second IPset file, *high_packet_tcp.set*, contains the external IPs for records with four or more packets.

The first set, for TCP traffic on 03/01/2003 can be generated with:

```
$ rwfilter --start-date=2003/03/01:00 --end-date=2003/03/01:23     \
       --proto=6 --packets=1-3 --pass=stdout                       \
  | rwset --sip-file=low_packet_tcp.set
```

The second set with:

```
$ rwfilter --start-date=2003/03/01:00 --end-date=2003/03/01:23    \
       --proto=6 --packets=4- --pass=stdout                       \
  | rwset --sip-file=high_packet_tcp.set
```

ENVIRONMENT

SILK_IPSET_RECORD_VERSION

This environment variable is used as the value for the **--record-version** when that switch is not provided. *Since SiLK 3.7.0.*

SILK_IPV6_POLICY

This environment variable is used as the value for **--ipv6-policy** when that switch is not provided.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_COMPRESSION_METHOD

This environment variable is used as the value for **--compression-method** when that switch is not provided. *Since SiLK 3.13.0.*

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwset** may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **rwset** may use this environment variable. See the FILES section for details.

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

SEE ALSO

rwsetbuild(1), **rwsetcat(1)**, **rwsettool(1)**, **rwsetmember(1)**, **rwfilter(1)**, **rwfileinfo(1)**, **rwbag(1)**, **rwuniq(1)**, **silk(7)**, **zlib(3)**

NOTES

Prior to SiLK 3.0, an IPset file could not contain IPv6 addresses and the record version was 2. The **--record-version** switch was added in SiLK 3.0 and its default was 3. In SiLK 3.6, an argument of 0 was allowed and made the default. Version 4 was added in SiLK 3.7 as was support for the SILK_IPSET_RECORD_VERSION environment variable. Version 5 was added in SiLK 3.14.

rwsetbuild

Create a binary IPset file from list of IPs

SYNOPSIS

```
rwsetbuild [{--ip-ranges | --ip-ranges=DELIM}]
      [--record-version=VERSION] [--invocation-strip]
      [--note-add=TEXT] [--note-file-add=FILENAME]
      [--compression-method=COMP_METHOD]
      [{INPUT_TEXT_FILE | -} [{OUTPUT_SET_FILE | -}]]

rwsetbuild --help

rwsetbuild --version
```

DESCRIPTION

rwsetbuild creates a binary IPset file from textual input. The IPset is written to the second command line argument if it has been specified; otherwise the IPset is written to the standard output if the standard output is not a terminal. **rwsetbuild** will not overwrite an existing file unless the SILK_CLOBBER environment variable is set. The textual input is read from the first command line argument if it has been specified; otherwise the text is read from the standard input if the standard input is not a terminal. A input file name of `stdin` or `-` means the standard input; an output file name of `stdout` or `-` means the standard output. **rwsetbuild** will read textual IPs from the terminal if the standard input is explicitly given as the input. **rwsetbuild** exits with an error if the input file cannot be read or the output file cannot be written.

Comments are ignored in the input file; they begin with the '`#`' symbol and continue to the end of the line. Whitespace and blank lines are also ignored. Otherwise, a line should contain a single IP addresses unless the **--ip-ranges** switch is specified, in which case a line may contain two IP addresses separated by the user-specified delimiter, which defaults to hyphen (`-`).

rwsetbuild supports IPv4 addresses and, when SiLK has been built with IPv6 support, IPv6 addresses. When the input contains a mixture of IPv4 and IPv6 addresses, the IPv4 addresses are mapped into the ::ffff:0:0/96 block of IPv6. When writing the IPset, **rwsetbuild** converts the output to IPv4 if all IPv6 addresses are in the ::ffff:0:0/96 block. **rwsetbuild** does not allow the input to contain both integer values and IPv6 addresses.

Each IP address must be expressed in one of these formats:

- Canonical IPv4 address (i.e., dotted decimal---all 4 octets are required):

  ```
  10.1.2.4
  ```

- An unsigned 32-bit integer:

  ```
  167838212
  ```

- Canonical IPv6 address:

```
2001:db8::f00
```

- Any of the above with a CIDR designation:

```
10.1.2.4/31
167838212/31
192.168.0.0/16
2001:db8::/48
```

- SiLK IP Wildcard: An IP Wildcard can represent multiple IPv4 or IPv6 addresses. An IP Wildcard contains an IP in its canonical form, except each part of the IP (where *part* is an octet for IPv4 or a hexadectet for IPv6) may be a single value, a range, a comma separated list of values and ranges, or the letter `x` to signify all values for that part of the IP (that is, `0-255` for IPv4). You may not specify a CIDR suffix when using IP Wildcard notation. IP Wildcard notation is not supported when the **--ip-ranges** switch is specified.

```
10.x.1-2.4,5
2001:db8::aaab-ffff,aaaa,0-aaa9
```

- IP Range: An IPv4 address, an unsigned 32-bit integer, or an IPv6 address to use as the start of the range, a delimiter, and an IPv4 address, an unsigned 32-bit integer, or an IPv6 address to use as the end of the range. The default delimiter is the hyphen (`-`), but a different delimiter may be specified as a parameter to the **--ip-ranges** switch. Whitespace around the IP addresses is ignored. Only valid when **--ip-ranges** is specified.

```
10.1.2.4-10.1.2.5
167838212-167838213
192.168.0.0-192.168.255.255
2001:db8::f00-2001:db8::fff
```

If an IP address cannot be parsed, **rwsetbuild** exits with an error.

Use **rwsetcat(1)** to see the contents of an IPset file. To check for a specific IP address in an IPset, use **rwsetmember(1)**. **rwsettool(1)** manipulates IPset files. To build an IPset file from SiLK Flow data, use **rwset(1)**.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--ip-ranges

--ip-ranges=*DELIM*

Allow lines of the the input file to contain a pair of IP addresses, separated by *DELIM*, that create an IP address range, and do not allow the IP Wildcard syntax. A line may also contain a single IP address or a 32-bit integer; these lines may have a CIDR designation. CIDR designations are not supported on lines that contain a pair of IP addresses. If *DELIM* is not specified, hyphen ('-') is used as the delimiter. When *DELIM* is a whitespace character, any amount of whitespace may surround and separate the two IP addresses. Since '`#`' is used to denote comments and newline is used to denote records, neither is a valid delimiter character.

--record-version=*VERSION*

Specify the format of the IPset records that are written to the output. *VERSION* may be 2, 3, 4, 5 or the special value 0. When the switch is not provided, the SILK_IPSET_RECORD_VERSION environment variable is checked for a version. The default version is 0.

0

Use the default version for an IPv4 IPset and an IPv6 IPset. Use the **--help** switch to see the versions used for your SiLK installation.

2

Create a file that may hold only IPv4 addresses and is readable by all versions of SiLK.

3

Create a file that may hold IPv4 or IPv6 addresses and is readable by SiLK 3.0 and later.

4

Create a file that may hold IPv4 or IPv6 addresses and is readable by SiLK 3.7 and later. These files are more compact that version 3 and often more compact than version 2.

5

Create a file that may hold only IPv6 addresses and is readable by SiLK 3.14 and later. When this version is specified, IPsets containing only IPv4 addresses are written in version 4. These files are usually more compact that version 4.

--invocation-strip

Do not record any command line history; that is, do not record the current command line invocation in the output file.

--note-add=*TEXT*

Add the specified *TEXT* to the header of the output file as an annotation. This switch may be repeated to add multiple annotations to a file. To view the annotations, use the **rwfileinfo(1)** tool.

--note-file-add=*FILENAME*

Open *FILENAME* and add the contents of that file to the header of the output file as an annotation. This switch may be repeated to add multiple annotations. Currently the application makes no effort to ensure that *FILENAME* contains text; be careful that you do not attempt to add a SiLK data file as an annotation.

--compression-method=*COMP_METHOD*

Specify the compression library to use when writing output files. If this switch is not given, the value in the SILK_COMPRESSION_METHOD environment variable is used if the value names an available compression method. When no compression method is specified, output to the standard output or to named pipes is not compressed, and output to files is compressed using the default chosen when SiLK was compiled. The valid values for *COMP_METHOD* are determined by which external libraries were found when SiLK was compiled. To see the available compression methods and the default method, use the **--help** or **--version** switch. SiLK can support the following *COMP_METHOD* values when the required libraries are available.

none

Do not compress the output using an external library.

zlib

Use the **zlib(3)** library for compressing the output, and always compress the output regardless of the destination. Using zlib produces the smallest output files at the cost of speed.

lzo1x

Use the *lzo1x* algorithm from the LZO real time compression library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead.

snappy

Use the *snappy* library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead. *Since SiLK 3.13.0.*

best

Use lzo1x if available, otherwise use snappy if available, otherwise use zlib if available. Only compress the output when writing to a file.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLE

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line.

Reading from a file:

```
$ echo 10.x.x.x > ten.txt
$ rwsetbuild ten.txt ten.set

$ echo 10.0.0.0/8 > ten.txt
$ rwsetbuild ten.txt ten.set

$ echo 10.0.0.0-10.255.255.255 > ten.txt
$ rwsetbuild --ip-ranges ten.txt ten.set

$ echo '167772160,184549375' > ten.txt
$ rwsetbuild --ip-ranges=, ten.txt ten.set
```

Reading from the standard input:

```
$ echo 192.168.x.x | rwsetbuild stdin private.set
```

Example input to **rwsetbuild**:

```
# A single address
10.1.2.4
# Two addresses in the same subnet
10.1.2.4,5
# The same two addresses
```

```
10.1.2.4/31
# The same two addresses
167838212/31
# A whole subnet
10.1.2.0-255
# The same whole subnet
10.1.2.x
# The same whole subnet yet again
10.1.2.0/24
# All RFC1918 space
10.0.0.0/8
172.16.0.0/12
192.168.0.0/16
# All RFC1918 space
10.x.x.x
172.16-20,21,22-31.x.x
192.168.x.x
# All RFC1918 space
167772160/8
2886729728/12
3232235520/16
# Everything ending in 255
x.x.x.255
# All addresses that end in 1-10
x.x.x.1-10
```

ENVIRONMENT

SILK_IPSET_RECORD_VERSION

This environment variable is used as the value for the **--record-version** when that switch is not provided. *Since SiLK 3.7.0.*

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_COMPRESSION_METHOD

This environment variable is used as the value for **--compression-method** when that switch is not provided. *Since SiLK 3.13.0.*

SEE ALSO

rwset(1), **rwsetcat(1)**, **rwsetmember(1)**, **rwsettool(1)**, **rwfileinfo(1)**, **silk(7)**, **zlib(3)**

NOTES

Prior to SiLK 3.0, an IPset file could not contain IPv6 addresses and the record version was 2. The **--record-version** switch was added in SiLK 3.0 and its default was 3. In SiLK 3.6, an argument of 0 was allowed and made the default. Version 4 was added in SiLK 3.7 as was support for the SILK_IPSET_RECORD_VERSION environment variable. Version 5 was added in SiLK 3.14.

rwsetcat

Print the IP addresses in a binary IPset file

SYNOPSIS

```
rwsetcat [--count-ips] [--print-statistics] [--print-ips]
      [{ --cidr-blocks | --cidr-blocks=0 | --cidr-blocks=1
         | --network-structure | --network-structure=STRUCTURE
         | --ip-ranges }]
      [--ip-format=FORMAT] [--integer-ips] [--zero-pad-ips]
      [--no-columns] [--column-separator=C] [--no-final-delimiter]
      [{--delimited | --delimited=C}]
      [--print-filenames | --print-filenames=0 | --print-filenames=1]
      [--output-path=PATH] [--pager=PAGER_PROG] [SET_FILE...]


rwsetcat --help


rwsetcat --version
```

DESCRIPTION

When run with no switches, **rwsetcat** reads each IPset file given on the command line and prints its constituent IP addresses to the standard output. If no file names are listed on the command line, **rwsetcat** attempts to read an IPset from the standard input.

By default, an IPset containing only IPv4 addresses is printed with one IP address per line. For other IPsets, **rwsetcat** uses CIDR blocks when printing. The **--cidr-blocks** switch may be used to choose which representation is used.

When processing a mix of IPv4 and IPv6 addresses, the SiLK tools that build IPsets (e.g., **rwset(1)**, **rwsetbuild(1)**) map each IPv4 address into the ::ffff:0:0/96 IPv6 netblock. For example 192.0.2.1 becomes ::ffff:192.0.2.1 or ::ffff:c000:201. For releases prior to SiLK 3.17.0, **rwsetcat** always displayed these addresses as IPv6 in the ::ffff:0:0/96 netblock. Starting in SiLK 3.17.0, **rwsetcat** shows these addresses as IPv4 unless the `map-v4` argument is given to the **--ip-format** switch.

rwsetcat can produce additional information about IPset files, such as the number of IPs they contain (use **--count**), the number of IPs in netblocks of arbitrary size (**--network-structure**), and the minimum and maximum IPs (**--print-statistics**).

To create an IPset file from SiLK Flow records, use **rwset(1)**. **rwsetbuild(1)** creates an IPset from textual input. An IPset may also be created by the **--coverset** switch on **rwbagtool(1)** and the **--to-ipset** switch of **rwaggbagtool(1)**. To determine whether an IPset file contains an IP address, use **rwsetmember(1)**.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--count-ips

Print a count of the number of IP addresses in the IPset file. This switch disables the printing of the IP addresses in the IPset file. Use **--print-ips** to print the contents of the IPset in addition to the count. When **--count-ips** is specified and more than one IPset file is provided, **rwsetcat** prepends the name of the input file and a colon to the IP address count. See the description of the **--print-filenames** switch for more information.

--print-statistics

Print a summary of the IPset. The summary includes the minimum IP address, the maximum IP address, the number of IP addresses in the IPset, and the number of IPs in a specific set of netblocks. For an IPset containing only IPv4 addresses, the netblocks are /8, /16, /24, and /27, and the output includes what percentage of IPv4 address space is covered. For an IPv6 IPset, the netblock are /8, /16, /24, /32, /40, /48, /56, /64, /72, /80, /88, /96, /112, and /120.

This switch disables the printing of the IP addresses in the IPset. Use **--print-ips** to print the contents of the IPset in addition to the statistics. When **--print-statistics** is specified and more than one IPset file is provided, **rwsetcat** prints the name of the input file, a colon, and a newline prior to printing the statistics. See the description of the **--print-filenames** switch for more information.

--print-ips

Force printing of the IP addresses, even when the **--count-ips** or **--print-statistics** option is provided.

--cidr-blocks

--cidr-blocks=0

--cidr-blocks=1

When an argument is not provided to the switch or when the argument is 1, group sequential IPs into the largest possible CIDR block and print CIDR blocks in the IPset file, If the argument is 0, print the individual IPs in the IPset file. By default, **rwsetcat** prints individual IPs for IPv4 IPsets and CIDR blocks for IPv6 IPsets. This switch may not be combined with the **--ip-ranges** or **--network-structure** switches.

--ip-ranges

Print the IPset in three pipe-delimited (|) columns where each row represents a contiguous IP range: the first column is the number of IPs in the range, the second is the start of the range, and the final is the end of the range. This prints the IPset in the fewest number of lines. This switch may not be combined with the **--cidr-blocks** or **--network-structure** switches.

--network-structure

--network-structure=*STRUCTURE*

For each numeric value in *STRUCTURE*, group the IPs in the IPset into a netblock of that size and print the number of hosts and, optionally, print the number of smaller, occupied netblocks that each larger netblock contains. When *STRUCTURE* begins with `v6:`, the IPs in the IPset are treated as IPv6 addresses, and any IPv4 addresses are mapped into the ::ffff:0:0/96 netblock. Otherwise, the IPs are treated as IPv4 addresses, and any IPv6 address outside the ::ffff:0:0/96 netblock is ignored. Aside from the initial `v6:` (or `v4:`, for consistency), *STRUCTURE* has one of following forms:

1. *NETBLOCK_LIST*/*SUMMARY_LIST*. Group IPs into the sizes specified in either *NETBLOCK_LIST* or *SUMMARY_LIST*. **rwsetcat** prints a row for each occupied netblock specified in *NETBLOCK_LIST*, where the row lists the base IP of the netblock, the number of hosts, and the number of smaller, occupied netblocks having a size that appears in either *NETBLOCK_LIST* or *SUMMARY_LIST*. (The values in *SUMMARY_LIST* are only summarized; they are not printed.)

2. *NETBLOCK_LIST*`/`. Similar to the first form, except all occupied netblocks are printed, and there are no netblocks that are only summarized.
3. *NETBLOCK_LIST***S**. When the character `S` appears anywhere in the *NETBLOCK_LIST*, **rwsetcat** provides a default value for the *SUMMARY_LIST*. That default is 8,16,24,27 for IPv4, and 48,64 for IPv6. **rwsetcat** ignores `S` if `/` is present.
4. *NETBLOCK_LIST*. When neither `S` nor `/` appear in *STRUCTURE*, the output does not include the number of smaller, occupied netblocks.
5. Empty. When *STRUCTURE* is empty or only contains `v6:` or `v4:`, the *NETBLOCK_LIST* prints a single row for the total network (the /0 netblock) giving the number of hosts and the number of smaller, occupied netblocks using the same default list specified in form 3.

NETBLOCK_LIST and *SUMMARY_LIST* contain a comma separated list of numbers between 0 (the total network) and the size for an individual host (32 for IPv4 or 128 for IPv6). The characters `T` and `H` may be used as aliases for 0 and the host netblock, respectively. In addition, when parsing the lists as IPv4 netblocks, the characters `A`, `B`, `C`, and `X` are supported as aliases for 8, 16, 24, and 27, respectively. A comma is not required between adjacent letters. The **--network-structure** switch disables printing of the IPs in the IPset file; specify the `H` argument to the switch to print each individual IP address. This switch may not be combined with the **--cidr-blocks** or **--ip-ranges** switches.

--ip-format=*FORMAT*

Specify how IP addresses are printed, where *FORMAT* is a comma-separated list of the arguments described below. When this switch is not specified, the SILK_IP_FORMAT environment variable is checked for a value and that format is used if it is valid. The default *FORMAT* is `canonical`,`unmap-v6`. (The default presentation of IPv4 addresses in a mixed IPv4-IPv6 IPset changed in SiLK 3.17.0 as described above in DESCRIPTION.) *Since SiLK 3.7.0.*

canonical

Print IP addresses in the canonical format. For an IPv4 address, use dot-separated decimal (`192.0.2.1`). Also use dot-separated decimal for IPv4-mapped IPv6 addresses (addresses in the ::ffff:0:0/96 netblock) unless *FORMAT* includes `map-v4`. For other IPv6 addresses, use either colon-separated hexadecimal (`2001:db8::1`) or a mixed IPv4-IPv6 representation for IPv4-compatible IPv6 addresses (the ::/96 netblock other than ::/127, e.g., `::192.0.2.1`). When `map-v4` is part of the argument, use the mixed representation for IPv4-mapped IPv6 addresses (the ::ffff:0:0/96 netblock, e.g., `::ffff:192.0.2.1`).

no-mixed

Print IP addresses in the canonical format (`192.0.2.1` or `2001:db8::1`) but do not used the mixed IPv4-IPv6 representations. For example, use `::c000:201` instead of `::192.0.2.1`. When *FORMAT* includes `map-v4`, also use `::ffff:c000:201` instead of `::ffff:192.0.2.1`. *Since SiLK 3.17.0.*

decimal

Print IP addresses as integers in decimal format. For example, print `192.0.2.1` and `2001:db8::1` as `3221225985` and `42540766411282592856903984951653826561`, respectively. Values in the ::ffff:0:0/96 netblock are not converted to IPv4 unless `unmap-v6` is explicitly given.

hexadecimal

Print IP addresses as integers in hexadecimal format. For example, print `192.0.2.1` and `2001:db8::1` as `c00000201` and `20010db800000000000000000000001`, respectively. Values in the ::ffff:0:0/96 netblock are not converted to IPv4 unless `unmap-v6` is explicitly given. **Note:** This setting does not apply to CIDR prefix values which are printed as decimal.

zero-padded
Make all IP address strings contain the same number of characters by padding numbers with leading zeros. For example, print `192.0.2.1` and `2001:db8::1` as `192.000.002.001` and `2001:0db8:0000:0000:0000:0000:0000:0001`, respectively. For IPv6 addresses, this setting implies `no-mixed`, so that `::192.0.2.1` is printed as `0000:0000:0000:0000:0000:0000:c000:0201`. As of SiLK 3.17.0, may be combined with any of the above, including `decimal` and `hexadecimal`. IPv4-mapped IPv6 addresses are printed as IPv6 unless *FORMAT* also includes `map-v4`, `decimal`, or `hexadecimal`. As of SiLK 3.18.0, the values of CIDR prefix are also zero-padded.

The following arguments modify certain IP addresses prior to printing. These arguments may be combined with the above formats.

map-v4
When the IPset contains only IPv4 addresses, change all IPv4 addresses to IPv4-mapped IPv6 addresses (addresses in the ::ffff:0:0/96 netblock) prior to formatting. For an IPset containing IPv6 addresses, do not map addresses in the ::ffff:0:0/96 netblock to IPv4. *Since SiLK 3.17.0.*

unmap-v6
When the IPset contains IPv6 addresses, change any IPv4-mapped IPv6 addresses (addresses in the ::ffff:0:0/96 netblock) to IPv4 addresses prior to formatting. This argument is enabled by default for the `canonical` and `no-mixed` formats. *Since SiLK 3.17.0.*

The following argument is also available:

force-ipv6
Set *FORMAT* to `map-v4`,`no-mixed`.

--integer-ips
Print IP addresses as integers. This switch is equivalent to **--ip-format=decimal**, it is deprecated as of SiLK 3.7.0, and it will be removed in the SiLK 4.0 release.

--zero-pad-ips
Print IP addresses as fully-expanded, zero-padded values in their canonical form. This switch is equivalent to **--ip-format=zero-padded**, it is deprecated as of SiLK 3.7.0, and it will be removed in the SiLK 4.0 release.

--no-columns
Disable fixed-width columnar output when printing the output from the **--network-structure** or **--ip-ranges** switch.

--column-separator=*C*
Use specified character between columns produced by the **--network-structure** and **--ip-ranges** switches. This character is also used after the final column when **--ip-ranges** is specified. When this switch is not specified, the default of '|' is used.

--no-final-delimiter
Do not print the column separator after the final column in the output produced by **--ip-ranges**. Normally a delimiter is printed.

--delimited

--delimited=*C*
Run as if **--no-columns --no-final-delimiter --column-sep**=*C* had been specified. That is, disable fixed-width columnar output; if character *C* is provided, it is used as the delimiter between columns instead of the default '|'.

--print-filenames

--print-filenames=0

--print-filenames=1

If an argument is not provided to the switch or if the argument is 1, print the name of the IPset file prior to printing information about the IPset file regardless of the number of IPset files specified on the command line or the type of information to be printed. If the switch is provided and its argument is 0, suppress printing the name of the IPset file regardless of the number of IPset files or type of information. When the switch is not provided, **rwsetcat**'s behavior depends on the type of information to be printed and on the number of input IPset files: If multiple IPset files are provided and **--count-ips** or **--print-statistics** is given, **rwsetcat** prints the name of a file, a colon (**:**), a newline (unless **--count-ips** was specified), and the requested information; otherwise, **rwsetcat** does not print the file name.

--output-path=*PATH*

Write the textual output to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output (and bypass the paging program). If *PATH* names an existing file, **rwsetcat** exits with an error unless the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. If this switch is not given, the output is either sent to the pager or written to the standard output. *Since SiLK 3.15.0.*

--pager=*PAGER_PROG*

When output is to a terminal, invoke the program *PAGER_PROG* to view the output one screen full at a time. This switch overrides the SILK_PAGER environment variable, which in turn overrides the PAGER variable. If the **--output-path** switch is given or if the value of the pager is determined to be the empty string, no paging is performed and all output is written to the terminal.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (**$**) represents the shell prompt. Some input lines are split over multiple lines in order to improve readability, and a backslash (\) is used to indicate such lines.

The file *sample.set* contains an IPset of IPv4 addresses, and the file *set1-v6.set* contains an IPset of IPv6 addresses.

Producing simple output with an IPv4 IPset

By default, **rwsetcat** prints the contents of an IPset.

```
$ rwsetcat sample.set
10.1.2.250
10.1.2.251
10.1.2.252
```

```
10.1.2.253
10.1.2.254
10.1.2.255
10.1.3.0
10.1.3.1
10.1.3.2
10.1.3.3
10.1.3.4
```

Use the **--cidr-blocks** switch to print the contents in CIDR notation.

```
$ rwsetcat --cidr-blocks sample.set
10.1.2.250/31
10.1.2.252/30
10.1.3.0/30
10.1.3.4
```

Add the **--ip-format** switch to change how the IPs are presented. For text-based sorting, use the **--ip-format=zero-padded** switch to force three digits per octet.

```
$ rwsetcat --ip-format=zero-padded --cidr-blocks sample.set
010.001.002.250/31
010.001.002.252/30
010.001.003.000/30
010.001.003.004
```

For numerical sorting, print the IPs as integers.

```
$ rwsetcat --ip-format=decimal sample.set
167838458
167838459
167838460
167838461
167838462
167838463
167838464
167838465
167838466
167838467
167838468
```

Getting simple output for an IPv6 IPset

When printing an IPset containing IPv6 addresses, addresses are grouped into CIDR blocks by default.

```
$ rwsetcat set1-v6.set
2001:db8:0:5::/68
2001:db8:0:5:f000::/68
2001:db8:0:c::/67
```

```
2001:db8:0:c:4000::/66
2001:db8:0:f:8000::/65
2001:db8:0:11::/64
2001:db8:0:12::/63
2001:db8:0:14::/62
2001:db8:0:18::/61
2001:db8:0:20::/60
2001:db8:0:40::/59
```

Specify an argument of 0 to the **--cidr-blocks** switch to see the individual IPs.

```
$ rwsetcat --cidr-blocks=0 set1-v6.set | head -4
2001:db8:0:5::
2001:db8:0:5::1
2001:db8:0:5::2
2001:db8:0:5::3
```

Finding the number of IPs in an IPset

The **--count-ips** switch prints the number IPs in the IPset.

```
$ rwsetcat --count-ips sample.set
11

$ rwsetcat --count-ips set1-v6.set
118059162071741130342 4
```

The number of IPs may also be produced using the **--network-structure** switch as described below.

Viewing IP ranges

To see contiguous IPs printed as ranges, use the **--ip-ranges** switch. The output has three columns that contain the length of the range, its starting IP, and its ending IP.

```
$ rwsetcat --ip-ranges sample.set
        11|     10.1.2.250|       10.1.3.4|
```

Since contiguous but different-sized CIDR blocks can be combined into a single range, the **--ip-ranges** switch prints the IPset in the first number of rows.

Add the **--ip-format=decimal** switch to see contiguous IPs printed as ranges of integers.

```
$ rwsetcat --ip-ranges --ip-format=decimal sample.set
        11| 167838458| 167838468|
```

Use the **--delimited** switch to produce the same output as a list of comma separated values.

```
$ rwsetcat --ip-ranges --ip-format=decimal --delimited=, sample.set
11,167838458,167838468
```

The UNIX **cut(1)** tool can be used to remove the number of IPs in the range, so that the output only contains the starting and ending IPs.

```
$ rwsetcat --ip-ranges --ip-format=decimal --delimited=, sample.set \
    | cut -d"," -f2,3
167838458,167838468

$ rwsetcat --ip-ranges set1-v6.set | cut -d'|' -f2,3
         2001:db8:0:5::|         2001:db8::5:fff:ffff:ffff:ffff
    2001:db8:0:5:f000::|        2001:db8::5:ffff:ffff:ffff:ffff
         2001:db8:0:c::|        2001:db8::c:1fff:ffff:ffff:ffff
    2001:db8:0:c:4000::|        2001:db8::c:7fff:ffff:ffff:ffff
    2001:db8:0:f:8000::|        2001:db8::f:ffff:ffff:ffff:ffff
        2001:db8:0:11::|       2001:db8::2f:ffff:ffff:ffff:ffff
        2001:db8:0:40::|       2001:db8::5f:ffff:ffff:ffff:ffff
```

Reading an IPset from the standard input

rwsetcat will read the IPset file from the standard input when no file name is given on the command line. An IP address converter is created by having the input to **rwsetcat** be the output from **rwsetbuild(1)**.

```
$ echo 10.10.10.10 | rwsetbuild | rwsetcat --ip-format=decimal
168430090
```

To see the unique source and destination IP addresses in the SiLK Flow file *data.rw*, use **rwset(1)** to generate an IPset and send the output of **rwset** to the standard input of **rwsetcat**.

```
$ rwset --any-file=stdout data.rw | rwsetcat | head -4
10.4.52.235
10.5.231.251
10.9.77.117
10.11.88.88
```

Getting multiple types of output

To see the contents of the IPset and also get a count of IPs, use multiple options.

```
$ rwsetcat --count-ips --cidr-blocks sample.set
11
10.1.2.250/31
10.1.2.252/30
10.1.3.0/30
10.1.3.4
```

Working with multiple IPset files

When multiple IPset files are specified on the command line, **rwsetcat** prints the contents of each file one after the other.

```
$ rwsetcat --cidr-blocks=1  sample.set set1-v6.set
10.1.2.250/31
10.1.2.252/30
10.1.3.0/30
10.1.3.4
2001:db8:0:5::/68
2001:db8:0:5:f000::/68
2001:db8:0:c::/67
2001:db8:0:c:4000::/66
2001:db8:0:f:8000::/65
2001:db8:0:11::/64
2001:db8:0:12::/63
2001:db8:0:14::/62
2001:db8:0:18::/61
2001:db8:0:20::/60
2001:db8:0:40::/59
```

To print the union of multiple the IPset files, use **rwsettool(1)** to join the files and have **rwsetcat** print the result.

```
$ rwsettool --union set1-v6.set sample.set \
  | rwsetcat --cidr-blocks=1
10.1.2.250/127
10.1.2.252/126
10.1.3.0/126
10.1.3.4
2001:db8:0:5::/68
2001:db8:0:5:f000::/68
2001:db8:0:c::/67
2001:db8:0:c:4000::/66
2001:db8:0:f:8000::/65
2001:db8:0:11::/64
2001:db8:0:12::/63
2001:db8:0:14::/62
2001:db8:0:18::/61
2001:db8:0:20::/60
2001:db8:0:40::/59
```

When counting the IPs in multiple IPset files, **rwsetcat** prepends the file name and a colon to the count. (The **-** argument causes **rwsetcat** to read the standard input in addition to the named file.)

```
$ cat set1-v6.set | rwsetcat --count-ips sample.set -
sample.set:11
-:118059162071741130342 4
```

Provide an argument of **0** to **--print-filenames** to suppress printing of the input IPset file name.

```
$ cat set1-v6.set \
    | rwsetcat --count-ips --print-filenames=0 sample.set -
11
118059162071741130342 4
```

Use the **--print-filenames** switch to force **rwsetcat** to print the file name when only one IPset is given.

```
$ rwsetcat --count-ips --print-filenames sample.set
sample.set:11
```

The **--print-filenames** switch also causes **rwsetcat** to print the file name when it normally would not.

```
$ rwsetcat --ip-ranges --ip-format=decimal --print-filenames sample.set
sample.set:
        11| 167838458| 167838468|
```

Seeing which netblocks are occupied

The **--network-structure** switch counts and prints information about which netblocks are occupied. The default output when no argument is given to the switch is a single line.

```
$ rwsetcat --network sample.set
TOTAL| 11 hosts in 1 /8, 1 /16, 2 /24s, and 2 /27s
```

The default is equivalent to an argument of `TS`.

```
$ rwsetcat --network=TS sample.set
TOTAL| 11 hosts in 1 /8, 1 /16, 2 /24s, and 2 /27s
```

An argument of `T` suppresses the subnet counts, and the output is the number of IPs in the IPset.

```
$ rwsetcat --network=T sample.set
TOTAL| 11
```

The argument `T` is equivalent to the 0 netblock.

```
$ rwsetcat --network=0 sample.set
TOTAL| 11
```

The subnets represented by `S` are 8, 16, 24, and 27. A different set of subnets to summarize may be specified by giving those subnets after a slash:

```
$ rwsetcat --network=T/12,18,30 sample.set
TOTAL| 11 hosts in 1 /12, 1 /18, and 4 /30s
```

The presence of a slash causes **rwsetcat** to ignore `S`.

```
$ rwsetcat --network=TS/12,18 sample.set
TOTAL| 11 hosts in 1 /12 and 1 /18
```

Putting a number in front of the slash adds a row the output for each netblock of that size that is occupied.

```
$ rwsetcat --network=30T/12,18 sample.set
  10.1.2.248/30     | 2 hosts
  10.1.2.252/30     | 4 hosts
  10.1.3.0/30       | 4 hosts
  10.1.3.4/30       | 1 host
TOTAL               | 11 hosts in 1 /12, 1 /18, and 4 /30s
```

For each row, the number of smaller, occupied netblocks is printed.

```
$ rwsetcat --network=12,18/30 sample.set
    10.1.0.0/18       | 11 hosts in 4 /30s
  10.0.0.0/12         | 11 hosts in 1 /18 and 4 /30s
TOTAL                 | 11 hosts in 1 /12, 1 /18, and 4 /30s
```

Although no numbers are required to follow the slash, the argument must include the slash for **rwsetcat** to produce the counts for each subnet.

```
$ rwsetcat --network=16,24/ sample.set
  10.1.2.0/24       | 6 hosts
  10.1.3.0/24       | 5 hosts
10.1.0.0/16         | 11 hosts in 2 /24s
```

```
$ rwsetcat --network=16,24 sample.set
  10.1.2.0/24       | 6
  10.1.3.0/24       | 5
10.1.0.0/16         | 11
```

For historical reasons, `A`, `B`, `C`, and `X` are equivalent to the 8, 16, 24, and 27 netblocks.

```
$ rwsetcat --network=B,C sample.set
  10.1.2.0/24       | 6
  10.1.3.0/24       | 5
10.1.0.0/16         | 11
```

Adding an argument of `H` tells **rwsetcat** to print the hosts.

```
$ rwsetcat --network=ABCXHST sample.set
          10.1.2.250      |
          10.1.2.251      |
          10.1.2.252      |
          10.1.2.253      |
          10.1.2.254      |
          10.1.2.255      |
        10.1.2.224/27     | 6 hosts
      10.1.2.0/24         | 6 hosts in 1 /27
          10.1.3.0        |
          10.1.3.1        |
          10.1.3.2        |
          10.1.3.3        |
```

```
          10.1.3.4        |
        10.1.3.0/27       | 5 hosts
      10.1.3.0/24         | 5 hosts in 1 /27
    10.1.0.0/16           | 11 hosts in 2 /24s and 2 /27s
  10.0.0.0/8              | 11 hosts in 1 /16, 2 /24s, and 2 /27s
TOTAL                     | 11 hosts in 1 /8, 1 /16, 2 /24s, and 2 /27s
```

The **--network-structure** switch defaults to treating the input as an IPset containing only IPv4 addresses. The results when running it on the IPv6 IPset file *set1-v6.set* are odd.

```
$ rwsetcat --network=TS set1-v6.set
TOTAL| 0 hosts in 0 /8s, 0 /16s, 0 /24s, and 0 /27s
```

The `v6:` prefix is required for **rwsetcat** to treat the input as IPv6.

```
$ rwsetcat --network=v6:TS set1-v6.set
TOTAL| 1180591620717411303424 hosts in 1 /48 and 66 /64s
```

As shown in that example, when the `v6:` prefix is given, the `S` character represents the 48 and 64 netblocks. The characters `A`, `B`, `C`, and `X` are not allowed when treating the input as IPv6.

```
$ rwsetcat --network=v6:A set1-v6.set
rwsetcat: Invalid network-structure character 'A'
```

The `H` character still represents the hosts.

```
$ rwsetcat --network=v6:H set1-v6.set | head -4
2001:db8:0:5::|
2001:db8:0:5::1|
2001:db8:0:5::2|
2001:db8:0:5::3|
```

When processing an IPv4 IPset as though it is IPv6, the IPv4 hosts are mapped into the ::ffff:0:0/96 netblock. (This is similar to passing a value of `force` to the **--ipv6-policy** switch on tools such as **rwcut(1)**.)

```
$ rwsetcat --network=v6:96TS sample.set
  ::ffff:0.0.0.0/96    | 11 hosts
TOTAL                  | 11 hosts in 1 /48, 1 /64, and 1 /96
```

When the `v6:` prefix is not present and **--network-structure** is used on an IPset containing IPv6 addresses, only those addresses in the ::ffff:0:0/96 netblock are visible to **rwsetcat**. This is similar to giving the **--ipv6-policy** switch an argument of `asv4`.

```
$ rwsettool --union set1-v6.set sample.set | rwsetcat --network=v6:TS
TOTAL| 1180591620717411303435 hosts in 2 /48s and 67 /64s
```

```
$ rwsettool --union set1-v6.set sample.set | rwsetcat --network=TS
TOTAL| 11 hosts in 1 /8, 1 /16, 2 /24s, and 2 /27s
```

Seeing a summary of an IPset

Use **--print-statistics** to get a summary of the IPset file.

```
$ rwsetcat --print-statistics --print-filenames sample.set
sample.set:
Network Summary
        minimumIP = 10.1.2.250
        maximumIP = 10.1.3.4
                11 hosts (/32s),    0.000000% of 2^32
                 1 occupied /8,     0.390625% of 2^8
                 1 occupied /16,    0.001526% of 2^16
                 2 occupied /24s,   0.000012% of 2^24
                 2 occupied /27s,   0.000001% of 2^27
```

```
$ rwsetcat --print-statistics set1-v6.set
Network Summary
       minimumIP = 2001:db8:0:5::
       maximumIP = 2001:db8::5f:ffff:ffff:ffff:ffff
                                            1 occupied /8
                                            1 occupied /16
                                            1 occupied /24
                                            1 occupied /32
                                            1 occupied /40
                                            1 occupied /48
                                            1 occupied /56
                                           66 occupied /64s
                                        16384 occupied /72s
                                      4194304 occupied /80s
                                   1073741824 occupied /88s
                                 274877906944 occupied /96s
                               70368744177664 occupied /104s
                            18014398509481984 occupied /112s
                          4611686018427387904 occupied /120s
                  11805916207174113034 24 hosts (/128s)
```

ENVIRONMENT

SILK_IP_FORMAT

This environment variable is used as the value for **--ip-format** when that switch is not provided. *Since SiLK 3.11.0.*

SILK_PAGER

When set to a non-empty string, **rwsetcat** automatically invokes this program to display its output a screen at a time. If set to an empty string, **rwsetcat** does not automatically page its output.

PAGER

When set and SILK_PAGER is not set, **rwsetcat** automatically invokes this program to display its output a screen at a time.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SEE ALSO

rwset(1), **rwsetbuild(1)**, **rwsettool(1)**, **rwsetmember(1)**, **rwbagtool(1)**, **rwcut(1)**, **silk(7)**, **cut(1)**

rwsetmember

Determine whether IP address(es) are members of an IPset

SYNOPSIS

```
rwsetmember [--count] [--quiet] PATTERN [INPUT_SET [INPUT_SET...]]

rwsetmember --help

rwsetmember --version
```

DESCRIPTION

rwsetmember determines whether an IP address or pattern exists in one or more IPset files, printing the name of the IPset files that contain the IP and optionally counting the number of matches in each file. *PATTERN* can be a single IP address, a CIDR block, or an IP Wildcard expressed in the same form as accepted by **rwsetbuild(1)**.

If an *INPUT_SET* is not given on the command line, **rwsetmember** will attempt to read an IPset from the standard input. To read the standard input in addition to the named files, use `-` or `stdin` as a file name. If an input file name ends in `.gz`, the file will be uncompressed as it is read.

When **rwsetmember** encounters an *INPUT_SET* file that it cannot read as an IPset, it prints an error message and moves to the next *INPUT_SET* file.

To create an IPset file from SiLK Flow records, use **rwset(1)**, and to create one from text, use **rwsetbuild(1)**. **rwsetcat(1)** prints an IPset file as text.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--count

Follow each set filename by a colon character and the number of pattern matches in the IPset. Files that do not match will still be printed, but with a zero match count. The **--count** switch is ignored when **--quiet** is also specified.

--quiet

Produce no standard output. The exit status of the program (see below) should be checked to determine whether any files matched.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line.

To quickly check whether a single set file contains an address (check the exit status):

```
$ rwsetmember --quiet 192.168.1.1 file.set
```

To display which of several set files (if any) match a given IP address:

```
$ rwsetmember 192.168.1.1 *.set
```

To display the same, but with counts from each file:

```
$ rwsetmember --count 192.168.1.1 *.set
```

To find all sets that contain addresses in the 10.0.0.0/8 subnet:

```
$ rwsetmember 10.0.0.0/8 *.set
```

To find files containing any IP address that ends with a number between 1 and 10 (this will use a lot of memory):

```
$ rwsetmember x.x.x.1-10 *.set
```

EXIT STATUS

rwsetmember exits with status code 0 if any file matched the pattern or 1 if there were no matches across any files or if there was a fatal error with the input.

SEE ALSO

rwset(1), **rwsetbuild(1)**, **rwsetcat(1)**, **silk(7)**

rwsettool

Operate on IPset files to produce a new IPset

SYNOPSIS

```
rwsettool { --union | --intersect | --difference
            | --symmetric-difference
            | --sample {--size=SIZE | --ratio=RATIO} [--seed=SEED]
            | --mask=NET_BLOCK_SIZE | --fill-blocks=NET_BLOCK_SIZE }
      [--output-path=PATH [--modify-inplace [--backup-path=BACKUP]]]
      [--record-version=VERSION] [--invocation-strip]
      [--note-strip] [--note-add=TEXT] [--note-file-add=FILE]
      [--compression-method=COMP_METHOD] [INPUT_SET ...]

rwsettool --help

rwsettool --version
```

DESCRIPTION

rwsettool performs a single operation on one or more IPset file(s) to produce a new IPset file.

The operations that **rwsettool** provides are

union

The union (or addition) of two IPsets is the set of IP addresses that are members in either set.

intersection

The intersection of two IPsets is the set of IP addresses that are members of both sets.

difference

The difference (or relative complement) of two IPsets is the set of IP addresses that are members of the first set but not members of the second.

symmetric-difference

The symmetric difference (or disjunctive union) of two IPsets is the set of IP addresses that are members of either set but not members of both. This is the equivalent to the intersection of the IPsets subtracted from the union of the IPsets. It is also equivalent to computing the union of both relative complements (the first set from the second and the second set from the first).

sample

The set of IP addresses in an IPset is randomly selected to produce a subset.

mask

For each CIDR-block (or net-block) of a user-specified size in the IPset, the IP addresses that are members of that net-block are replaced by a single IP address at the start of the net-block. Empty net-blocks are not changed.

fill-blocks

For each CIDR-block (or net-block) of a user-specified size in the IPset, the IP addresses that are members of that net-block are extended so that every IP address in that net-block is a member of the set. Empty net-blocks are not changed.

More details are provided in the OPTIONS section.

rwsettool reads the IPsets specified on the command line; when no IPsets are listed, **rwsettool** attempts to read an IPset from the standard input. The strings `stdin` or `-` can be used as the name of an input file to force **rwsettool** to read from the standard input. The resulting IPset is written to the location specified by the **--output-path** switch or to the standard output if that switch is not provided. Using the strings `stdout` or `-` as the argument to **--output-path** causes **rwsettool** to write the IPset to the standard output. **rwsettool** exits with an error if an attempt is made to read an IPset from the terminal or write an IPset to the terminal.

In SiLK 3.21.0, **rwsettool** added the **--modify-inplace** switch which correctly handles the case when an input file is also used as the output file. That switch causes **rwsettool** to write the new IPset to a temporary file first and then replace the original output file. The **--backup-path** switch may be used in conjunction with **--modify-inplace** to set the pathname where the original output file is copied.

To create an IPset file from SiLK Flow records, use **rwset(1)**, and to create one from text, use **rwsetbuild(1)**. **rwsetcat(1)** prints an IPset file as text. To determine whether an IPset file contains an IP address, use **rwsetmember(1)**.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

Operation Switches

Exactly one of the following operation switches must be provided:

--union

Perform the set union operation: The resulting IPset contains each IP address that is a member of *any* of the input IPsets.

--intersect

Perform the set intersection operation: The resulting IPset contains each IP address that is a member of *all* of the input IPsets.

--difference

Perform the set difference operation: The resulting IPset contains each IP address that is a member of the first IPset and not a member of any subsequent IPsets.

--symmetric-difference

Perform the symmetric difference operation: For two input sets, the resulting IPset contains each IP address that is a member of one of the input IPsets but not both. For each additional IPset, **rwsettool** computes the symmetric difference of the current result with the additional IPset. For three input sets, the output IPset contains each IP address that is a member of either one of the IPsets or of all three IPsets. *Since SiLK 3.13.0.*

--sample

Select a random sample of IPs from the input IPsets. The size of the subset must be specified by either the **--size** or **--ratio** switches described next. In the case of multiple input IPsets, the resulting IPset is the union of all IP addresses sampled from **each** of the input IPsets. That is, each IPset is individually sampled, and the results are merged.

--size=*SIZE*

Create an IPset containing the union of randomly selecting exactly *SIZE* IP addresses from each input IPset. If the number of IP addresses in an input IPset is less than or equal to *SIZE*, all members of that IPset are included in the result. When the input sets are completely disjoint and each set has at least *SIZE* members, the number of IP addresses in the result is the product of *SIZE* and the number of inputs.

--ratio=*RATIO*

Create an IPset where the probability of including each IP address of each input IPset in the result is *RATIO*, specified as a floating point number between 0.0 and 1.0. For each input IP address, **rwsettool** computes a pseudo-random number between 0 and 1 and adds the IP address to the result when the number is less than *RATIO*. The exact size of the subset may vary with each invocation.

--seed=*SEED*

Seed the pseudo-random number generator with value *SEED*. By default, the seed varies for each invocation. Seeding with a specific value produces repeatable results given the same input sets.

--mask=*NET_BLOCK_SIZE*

Perform a (sparse) masking operation: The resulting IPset contains one IP address for each /*NET_BLOCK_SIZE* CIDR block in the input IPset(s) that contains one or more IP addresses in that CIDR block. That is, **rwsettool** visits each /*NET_BLOCK_SIZE* CIDR block in the IPset. If the block is empty, no change is made; otherwise the block is cleared (all IPs removed) and the lowest IP address in that block is made a member of the set. *NET_BLOCK_SIZE* should be value between 1 and 32 for IPv4 sets and between 1 and 128 for IPv6 sets. Contrast with **--fill-blocks**.

--fill-blocks=*NET_BLOCK_SIZE*

Perform a (non-sparse) masking operation: The resulting IPset contains a completely full /*NET_BLOCK_SIZE* block for each /*NET_BLOCK_SIZE* CIDR block in the input IPset(s) that contain one or more IP addresses in that CIDR block. That is, **rwsettool** visits each /*NET_BLOCK_SIZE* CIDR block in the IPset; if the block is empty, no change is made, otherwise all IP addresses in the block are made members of the set. *NET_BLOCK_SIZE* should be value between 1 and 32 for IPv4 sets and between 1 and 128 for IPv6 sets. Contrast with **--mask**.

Output Switches

These switches control the output:

--output-path=*PATH*

Write the resulting IPset to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output. If *PATH* names an existing file, **rwsettool** exits with an error unless the **--modify-inplace** switch is given or the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. If **--output-path** is not given, the output is written to the standard output. Attempting to write the binary output to a terminal causes **rwsettool** to exit with an error.

--modify-inplace

Allow **rwsettool** to overwrite an existing file and properly account for the output file (*PATH*) also being an input file. When this switch is given, **rwsettool** writes the output to a temporary location first, then overwrites *PATH*. **rwsettool** attempts to copy the permission, owner, and group from the original file to the new file. The switch is ignored when *PATH* does not exist or the output is the standard output or standard error. **rwsettool** exits with an error when this switch is given and *PATH* is not a regular file. If **rwsettool** encounters an error or is interrupted prior to closing the temporary file, the temporary file is removed. See also **--backup-path**. *Since SiLK 3.21.0.*

--backup-path=*BACKUP*

Move the file named by **--output-path** (*PATH*) to the path *BACKUP* immediately prior to moving the temporary file created by **--modify-inplace** over *PATH*. If *BACKUP* names a directory, the file is moved into that directory. This switch will overwrite an existing file. If *PATH* and *BACKUP* point to the same location, the output is written to *PATH* and no backup is created. If **BACKUP** cannot be created, the new IPset file is left in the temporary file and **rwsettool** exits with a message and an error. **rwsettool** exits with an error if this switch is given without **--modify-inplace**. *Since SiLK 3.21.0.*

--record-version=*VERSION*

Specify the format of the IPset records that are written to the output. *VERSION* may be 2, 3, 4, 5 or the special value 0. When the switch is not provided, the SILK_IPSET_RECORD_VERSION environment variable is checked for a version. The default version is 0.

0

Use the default version for an IPv4 IPset and an IPv6 IPset. Use the **--help** switch to see the versions used for your SiLK installation.

2

Create a file that may hold only IPv4 addresses and is readable by all versions of SiLK.

3

Create a file that may hold IPv4 or IPv6 addresses and is readable by SiLK 3.0 and later.

4

Create a file that may hold IPv4 or IPv6 addresses and is readable by SiLK 3.7 and later. These files are more compact that version 3 and often more compact than version 2.

5

Create a file that may hold only IPv6 addresses and is readable by SiLK 3.14 and later. When this version is specified, IPsets containing only IPv4 addresses are written in version 4. These files are usually more compact that version 4.

--invocation-strip

Do not record any command line history; that is, do not record the current command line invocation in the output. The invocation history of the input IPset files is always stripped.

--note-strip

Do not copy the notes (annotations) from the input files to the output file. Normally notes from the input files are copied to the output.

--note-add=*TEXT*

Add the specified *TEXT* to the header of the output file as an annotation. This switch may be repeated to add multiple annotations to a file. To view the annotations, use the **rwfileinfo(1)** tool.

--note-file-add=*FILENAME*

Open *FILENAME* and add the contents of that file to the header of the output file as an annotation. This switch may be repeated to add multiple annotations. Currently the application makes no effort to ensure that *FILENAME* contains text; be careful that you do not attempt to add a SiLK data file as an annotation.

--compression-method=*COMP_METHOD*

Specify the compression library to use when writing output files. If this switch is not given, the value in the SILK_COMPRESSION_METHOD environment variable is used if the value names an available compression method. When no compression method is specified, output to the standard output or to named pipes is not compressed, and output to files is compressed using the default chosen when SiLK was compiled. The valid values for *COMP_METHOD* are determined by which external libraries were found when SiLK was compiled. To see the available compression methods and the default method, use the **--help** or **--version** switch. SiLK can support the following *COMP_METHOD* values when the required libraries are available.

none

Do not compress the output using an external library.

zlib

Use the **zlib(3)** library for compressing the output, and always compress the output regardless of the destination. Using zlib produces the smallest output files at the cost of speed.

lzo1x

Use the *lzo1x* algorithm from the LZO real time compression library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead.

snappy

Use the *snappy* library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead. *Since SiLK 3.13.0.*

best

Use lzo1x if available, otherwise use snappy if available, otherwise use zlib if available. Only compress the output when writing to a file.

Additional Switches

rwsettool supports these additional switches:

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

Assume the following IPsets:

```
A.set = { 1, 2, 4, 6 }
B.set = { 1, 3, 5, 7 }
C.set = { 1, 3, 6, 8 }
D.set = { } (empty set)
```

Set Union Examples

The union of two IPsets contains the IP addresses that are members of either IPset. The union of multiple IPsets contains the IP addresses that are members of any of the sets. The resulting IPset does not depend on the order of the input IPsets. The union of a single IPset, of an IPset with itself, and of an IPset with an empty IPset is the original IPset.

OPTIONS	RESULT
--union A.set B.set	{ 1, 2, 3, 4, 5, 6, 7 }
--union A.set C.set	{ 1, 2, 3, 4, 6, 8 }
--union A.set B.set C.set	{ 1, 2, 3, 4, 5, 6, 7, 8 }
--union C.set D.set	{ 1, 3, 6, 8 }
--union A.set	{ 1, 2, 4, 6 }
--union A.set A.set	{ 1, 2, 4, 6 }

Set Intersection Examples

The intersection of two IPsets contains the IP addresses that are members of both IPsets (that is, the IP addresses they have in common). The intersection of multiple IPsets contains the IP addresses that are members of all of the sets. The resulting IPset does not depend on the order of the input IPsets. The intersection of a single IPset is the original IPset. The intersection of an IPset with itself is the original IPset. The intersection of an IPset with an empty IPset is an empty IPset.

OPTIONS	RESULT
--intersect A.set B.set	{ 1 }
--intersect A.set C.set	{ 1, 6 }
--intersect B.set C.set	{ 1, 3 }
--intersect A.set B.set C.set	{ 1 }
--intersect A.set D.set	{ }
--intersect A.set	{ 1, 2, 4, 6 }
--intersect A.set A.set	{ 1, 2, 4, 6 }

Set Difference Examples

The difference of two IPsets contains the IP addresses that are members of the first set but not members of the second. The difference of multiple IPsets contains the IP addresses in the first set that are not members of any other IPset. The resulting IPset is dependent on the order of the input IPsets. Using the difference operation on a single IPset gives that IPset. The difference of an IPset with an empty IPset is the first IPset. The difference of an IPset with itself is the empty IPset.

OPTIONS	RESULT
--difference A.set B.set	{ 2, 4, 6 }
--difference B.set A.set	{ 3, 5, 7 }
--difference A.set B.set C.set	{ 2, 4 }
--difference C.set B.set A.set	{ 8 }
--difference C.set D.set	{ 1, 3, 6, 8 }
--difference D.set C.set	{ }
--difference A.set	{ 1, 2, 4, 6 }
--difference A.set A.set	{ }

Set Symmetric Difference Examples

The symmetric difference (or relative complement) of two IPsets contains the IP addresses that are members of either set but not members of both sets. For each additional input IPset, **rwsettool** computes the symmetric difference of the current result with the that IPset. The resulting IPset contains the IP addresses that are members of an odd number of the input sets. The resulting IPset does not depend on the order of the input IPsets. Using the symmetric difference operation on a single IPset gives that IPset. The symmetric difference of an IPset with an empty IPset is the first IPset. The symmetric difference of an IPset with itself is the empty IPset.

OPTIONS	RESULT
--symmetric A.set B.set	{ 2, 3, 4, 5, 6, 7 }
--symmetric A.set C.set	{ 2, 3, 4, 8 }
--symmetric A.set D.set	{ 1, 2, 4, 6 }
--symmetric C.set B.set	{ 5, 6, 7, 8 }
--symmetric A.set B.set C.set	{ 1, 2, 4, 5, 7, 8 }
--symmetric A.set	{ 1, 2, 4, 6 }
--symmetric A.set A.set	{ }

Finding IP Addresses Unique to an Input Set

Using the symmetric difference on three or more IPsets **does not** result in an IPset containing the IP addresses that are members of a single input set. To compute that, use the Bag tools as follows.

1. First, use **rwbagbuild(1)** to create an empty bag file */tmp/b.bag*.

   ```
   $ echo "" | rwbagbuild --bag-input=stdin --output-path=/tmp/b.bag
   ```

2. For each input IPset, *i.set*, use **rwbagbuild** to create a bag from the IPset, and use **rwbagtool(1)** to add that bag to *b.bag*.

   ```
   $ rwbagbuild --set-input=i.set   \
     | rwbagtool --add - /tmp/b.bag --output-path=/tmp/b2.bag
   $ mv /tmp/b2.bag /tmp/b.bag
   ```

To do that in a loop, run

```
$ for i in *.set ; do \
        rwbagbuild --set-input=$i  \
        | rwbagtool --add - /tmp/b.bag --output-path=/tmp/b2.bag ; \
        mv /tmp/b2.bag /tmp/b.bag ; \
  done
```

3. Use **rwbagtool** to create a coverset named *unique.set* that contains the IP addresses in *b.bag* whose counter is 1.

```
$ rwbagtool --maxcounter=1 --coverset --output-path=unique.set \
        /tmp/b.bag
```

A different approach may be used which does not require temporary files. Use **rwsetcat(1)** to convert the IPset files to text and feed that data to **rwbagbuild**. (When **rwsetcat** is invoked on multiple IPset files, it prints the contents of each individual IPset file, and as **rwbagbuild** processes the text, it increments an IP address's counter each time the IP appears in the input.) Use **rwbagtool** to create the IPset as shown in Step 3 above.

```
$ rwsetcat --cidr-blocks=1 *.set   \
  | rwbagbuild --bag-input=-       \
  | rwbagtool  --maxcounter=1 --coverset --output=unique.set
```

Set Sampling Examples

The **--sample** switch creates a subset that contains IP addresses that have been randomly selected from the input IPset(s).

The **--size** switch selects exactly *SIZE* IP addresses from each input set, but the number of IP addresses in the result may be less than the product of *SIZE* and the number of inputs when the input sets have IPs in common or when an IPset has fewer than *SIZE* members.

When using the **--size** switch, the probability of selecting an individual IP address varies with the number of IPs to be selected and the number of IPs remaining in the set. If N is the number of IPs in a set, the probability of selecting the first IP is $SIZE/N$. If that IP is selected, the probability of selecting the second is ($SIZE$-1)/(N-1), but if the first IP is not selected, the probability of selecting the second is $SIZE/(N\text{-}1)$.

```
+---------------------------------+-----------------------------+
| COMMAND                         | RESULT                      |
+---------------------------------+-----------------------------+
| --sample --size 2 A.set         | { 1, 4 }                    |
| --sample --size 2 A.set         | { 1, 6 }                    |
| --sample --size 3 A.set         | { 2, 4, 6 }                 |
| --sample --size 2 A.set B.set   | { 1, 2, 5, 7 }              |
| --sample --size 2 A.set B.set   | { 3, 4, 5, 6 }              |
| --sample --size 2 A.set B.set   | { 1, 4, 5 }                 |
+---------------------------------+-----------------------------+
```

The argument to the **--ratio** switch is the probability of choosing an individual IP address. For each IP address in the input, the IP is added to the output when a pseudo-random number between 0 and 1 is less then the argument to **--ratio**. The number of IP addresses in the result varies with each invocation.

```
+----------------------------------+----------------------------+
| COMMAND                          | RESULT                     |
+----------------------------------+----------------------------+
| --sample --ratio 0.5 A.set       | { 2, 6 }                   |
| --sample --ratio 0.5 A.set       | { 4 }                      |
| --sample --ratio 0.5 A.set B.set | { 1, 3 }                   |
| --sample --ratio 0.5 A.set B.set | { 2, 3, 5, 6, 7 }          |
+----------------------------------+----------------------------+
```

Set Masking and Block-Filling Examples

The goal of the **--mask** and **--fill-blocks** switches is to produce an IPset whose members are on user-defined CIDR-block (or net-block) boundaries. (In some ways, these switches produce output that is similar to the **--network-structure** switch on **rwsetcat(1)**.)

The **--mask** and **--fill-blocks** switches require a decimal argument that is a CIDR-block network mask size. For example, the argument 24 represents 256 IPv4 addresses. **rwsettool** visits each block of that size in the input IPset. If no IP addresses appear in that block, the result also has no IPs in the block. If one or more IP addresses appear in that block, the output IPset has either the lowest address in that block as a member (for **--mask**) or all IP addresses in that block as members (for **--fill-blocks**.

For example, consider the IPset *s.set* containing the three IP addresses.

```
$ rwsetcat --cidr-blocks=1 s.set
10.1.1.1
10.1.1.2
10.1.3.1
```

Specifying **--mask=24** produces an IPset containing two IP addresses.

```
$ rwsettool --mask=24 s.set | rwsetcat --cidr-blocks=1
10.1.1.0
10.1.3.0
```

Specifying **--fill-blocks=24** produces an IPset containing 512 IP addresses.

```
$ rwsettool --fill-block=24 s.set | rwsetcat --cidr-blocks=1
10.1.1.0/24
10.1.3.0/24
```

Consider *t.set* that contains four IP addresses.

```
$ rwsetcat --cidr-blocks=1 t.set
10.1.1.1
10.1.1.2
10.1.2.5
10.1.3.1
```

Running **--mask=24** and **--fill-blocks=24** on that file produces the following.

```
$ rwsettool --mask=24 t.set | rwsetcat --cidr-blocks=1
10.1.1.0
10.1.2.0
10.1.3.0


$ rwsettool --fill-block=24 t.set | rwsetcat --cidr-blocks=1
10.1.1.0/24
10.1.2.0/23
```

rwsetcat merges 10.1.2.0/24 and 10.1.3.0.24 into a single /23.

When multiple IPsets are specified on the command line, the union of the IPsets is computed prior to performing the mask or fill-blocks operation. The result is not dependent on the order of the IPsets.

Mixed IPv4 and IPv6 Examples

Suppose the IPset file *mixed.set* contains IPv4 and IPv6 addresses. To create an IPset file that contains only the IPv4 addresses, intersect *mixed.set* with the IPset *all-v4.set*, which is an IPset that contains all of IPv4 space (::ffff:0:0/96).

```
$ echo '::ffff:0:0/96' | rwsetbuild - all-v4.set


$ rwsettool --intersect mixed.set all-v4.set > subset-v4.set
```

To create an IPset file that contains only the IPv6 addresses, subtract *all-v4.set* from *mixed.set*:

```
$ rwsettool --difference mixed.set all-v4.set > subset-v6.set
```

The previous two commands may also be performed without having to write create the *all-v4.set* IPset file.

```
$ echo '::ffff:0:0/96'   \
  | rwsettool --intersect mixed.set - > subset-v4.set


$ echo '::ffff:0:0/96'   \
  | rwsettool --difference mixed.set - > subset-v6.set
```

Comparing Two IPsets Example

To determine if two IPset files contain the same set of IP addresses, use the **--symmetric-difference** switch and then count the number of IP addresses of the result with **rwsetcat**. If the count is 0, the files contain the same IP addresses.

```
$ cp A.set A2.set
$ rwsettool --symmetric-difference A.set A2.set  \
  | rwsetcat --count
0
```

Changing a File's Format

To share an IPset file with a user who has an older version of SiLK that includes different compression libraries, it may be necessary to change the the record-version or the compression-method of an IPset file.

It is not possible to change those aspects of the file directly. A new file must be created first, and then you may then replace the old file with the new file.

To create a new file that uses a different record-version or compression-method of the IPset file *A.set*, use **rwsettool** with the **--union** switch and specify the desired arguments:

```
$ rwsettool --union --record-version=5 --output-path=A2.set A.set

$ rwsettool --union --compression=none --output-path=A3.set A.set

$ rwsettool --union --record-version=2 --compression=best \
       --output-path=A4.set A.set
```

ENVIRONMENT

SILK_IPSET_RECORD_VERSION

This environment variable is used as the value for the **--record-version** when that switch is not provided. *Since SiLK 3.7.0.*

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_COMPRESSION_METHOD

This environment variable is used as the value for **--compression-method** when that switch is not provided. *Since SiLK 3.13.0.*

SEE ALSO

rwset(1), **rwsetbuild(1)**, **rwsetcat(1)**, **rwsetmember(1)**, **rwbagbuild(1)**, **rwbagtool(1)**, **rwfileinfo(1)**, **silk(7)**, **zlib(3)**

NOTES

Prior to SiLK 3.0, an IPset file could not contain IPv6 addresses and the record version was 2. The **--record-version** switch was added in SiLK 3.0 and its default was 3. In SiLK 3.6, an argument of 0 was allowed and made the default. Version 4 was added in SiLK 3.7 as was support for the SILK_IPSET_RECORD_VERSION environment variable. Version 5 was added in SiLK 3.14.

The **--modify-inplace** switch was added in SiLK 3.21. When **--backup-path** is also given, there is a small time window when the original file does not exist: the time between moving the original file to the backup location and moving the temporary file into place.

rwsilk2ipfix

Convert SiLK Flow records to IPFIX records

SYNOPSIS

```
rwsilk2ipfix [--ipfix-output=PATH] [--no-site-name-elements]
      [--print-statistics] [--single-template]
      [--site-config-file=FILENAME]
      {[--xargs] | [--xargs=FILENAME] | [FILE [FILE ...]]}

rwsilk2ipfix --help

rwsilk2ipfix --version
```

DESCRIPTION

rwsilk2ipfix reads SiLK Flow records, converts the records to an IPFIX (Internet Protocol Flow Information eXport) format, and writes the IPFIX records to the path specified by **--ipfix-output** or to the standard output when the **--ipfix-output** switch is not provided and standard output is not the terminal.

rwsilk2ipfix reads SiLK Flow records from the files named on the command line or from the standard input when no file names are specified and **--xargs** is not present. To read the standard input in addition to the named files, use `-` or `stdin` as a file name. If an input file name ends in `.gz`, the file is uncompressed as it is read. When the **--xargs** switch is provided, **rwsilk2ipfix** reads the names of the files to process from the named text file or from the standard input if no file name argument is provided to the switch. The input to **--xargs** must contain one file name per line.

The IPFIX records generated by **rwsilk2ipfix** will contain ten information elements that are in the Private Enterprise space for CERT (the IPFIX Private Enterprise Number of CERT is 6871). These ten information elements fall into three groups:

- Elements 30 and 31 contain the packing information that was determined by **rwflowpack(8)**, specifically the flowtype and the sensor. These values correspond to numbers specified in the site configuration file, **silk.conf(5)**.
- Elements 938, 939, 940, and 941 contain the names that correspond to the values in elements 30 and 31. These elements are not exported if **rwsilk2ipfix** is unable to find the *silk.conf* file or if the **--no-site-name-elements** switch is provided. *Since SiLK 3.20.0.*
- Elements 14, 15, 32, and 33 contain information elements generated by the **yaf(1)** flow meter (http://tools.netsa.cert.org/yaf/). The information elements may be present even if **yaf** was not used to generate the flow records, but their value will be empty or 0.

For each of the ten information elements that **rwsilk2ipfix** produces, the following table lists its numeric ID, its length in octets (or `v` for variable length), its name, the field name it corresponds to on **rwcut(1)**, and a brief description.

```
30  1  silkFlowtypeId    class & type   How rwflowpack categorized
                                        the flow record
```

```
 31  2  silkSensorId      sensor         Sensor where the flow was
                                        collected
938  v  silkFlowtypeName  -              Name of the silkFlowtypeId
                                        as read from F<silk.conf>
939  v  silkClassName     class          Class name derived from
                                        the silkFlowtypeId
940  v  silkTypeName      type           Type name derived from the
                                        silkFlowtypeId
941  v  silkSensorName    sensor         Name of the silkSensorId
                                        as read from F<silk.conf>
 14  1  initialTCPFlags   initialFlags   TCP flags on first packet in
                                        the flow record
 15  1  unionTCPFlags     sessionFlags   TCP flags on all packets in
                                        the flow except the first
 32  1  silkTCPState      attributes     Flow continuation attributes
                                        set by generator
 33  2  silkAppLabel      application    Guess by flow generator as
                                        to the content of traffic
```

Note: Elements 30 and 31, silkFlowtypeId and silkSensorId, may appear as silkFlowType and silkFlowSensor in some documentation.

Templates

Since SiLK 3.12.0, **rwsilk2ipfix** has used ten different IPFIX templates for writing SiLK Flow records, and the format of these templates changed in SiLK 3.20.0. The **--single-template** switch causes **rwsilk2ipfix** to revert to its previous behavior and use a single template for all records.

The final four elements in these templates (Elements 938--941) are variable length and they are not included if the site configuration file (**silk.conf(5)**) is not available or if the **--no-site-name-elements** option is given.

1. Template ID 0x9DD0 (40400), for IPv4 records whose protocol is not ICMP, ICMPv6, UDP, SCTP, or TCP:

```
 OCTETS    INFORMATION ELEMENT (PEN, ID)    POS    SILK FIELD
=======    =============================    ===    =============
  0-  7    flowStartMilliseconds (152)        1    sTime
  8- 15    flowEndMilliseconds (153)          2    sTime + duration
 16- 19    packetDeltaCount (2)               3    packets
 20- 23    octetDeltaCount (1)                4    bytes
 24- 25    ingressInterface (10)              5    in
 26- 27    egressInterface (14)               6    out
 28- 29    silkAppLabel (6871, 33)            7    application
 30- 31    silkSensorId (6871, 31)            8    sensor
 32        silkFlowtypeId (6871, 30)          9    class & type
 33        silkTCPState (6871, 32)           10    attributes
 34        protocolIdentifier (4)            11    protocol
 35- 38    sourceIPv4Address (8)             12    sIP
 39- 42    destinationIPv4Address (12)       13    dIP
 43- 46    ipNextHopIPv4Address (15)         14    nhIP
```

```
 47-  v    silkFlowtypeName (6871, 938)      15    class & type
  v-  v    silkClassName (6871, 939)         16    class
  v-  v    silkTypeName (6871, 940)          17    type
  v-  v    silkSensorName (6871, 941)        18    sensor
```

2. Template ID 0x9DD1 (40401), for ICMP IPv4 records:

```
 OCTETS    INFORMATION ELEMENT (PEN, ID)    POS    SILK FIELD
=======    =============================    ===    =============
  0-  7    flowStartMilliseconds (152)        1    sTime
  8- 15    flowEndMilliseconds (153)          2    sTime + duration
 16- 19    packetDeltaCount (2)               3    packets
 20- 23    octetDeltaCount (1)                4    bytes
 24- 25    ingressInterface (10)              5    in
 26- 27    egressInterface (14)               6    out
 28- 29    silkAppLabel (6871, 33)            7    application
 30- 31    silkSensorId (6871, 31)            8    sensor
 32        silkFlowtypeId (6871, 30)          9    class & type
 33        silkTCPState (6871, 32)           10    attributes
 34        protocolIdentifier (4)            11    protocol
 35- 36    icmpTypeCodeIPv4                  12    dPort
 37- 40    sourceIPv4Address (8)             13    sIP
 41- 44    destinationIPv4Address (12)       14    dIP
 45- 48    ipNextHopIPv4Address (15)         15    nhIP
 49-  v    silkFlowtypeName (6871, 938)       16    class & type
  v-  v    silkClassName (6871, 939)         17    class
  v-  v    silkTypeName (6871, 940)          18    type
  v-  v    silkSensorName (6871, 941)        19    sensor
```

3. Template ID 0x9DD2 (40402), for IPv4 records whose protocol is UDP or SCTP:

```
 OCTETS    INFORMATION ELEMENT (PEN, ID)    POS    SILK FIELD
=======    =============================    ===    =============
  0-  7    flowStartMilliseconds (152)        1    sTime
  8- 15    flowEndMilliseconds (153)          2    sTime + duration
 16- 19    packetDeltaCount (2)               3    packets
 20- 23    octetDeltaCount (1)                4    bytes
 24- 25    ingressInterface (10)              5    in
 26- 27    egressInterface (14)               6    out
 28- 29    silkAppLabel (6871, 33)            7    application
 30- 31    silkSensorId (6871, 31)            8    sensor
 32        silkFlowtypeId (6871, 30)          9    class & type
 33        silkTCPState (6871, 32)           10    attributes
 34        protocolIdentifier (4)            11    protocol
 35- 36    sourceTransportPort (7)           12    sPort
 37- 38    destinationTransportPort (11)     13    dPort
 39- 42    sourceIPv4Address (8)             14    sIP
 43- 46    destinationIPv4Address (12)       15    sIP
 47- 50    ipNextHopIPv4Address (15)         16    nhIP
 51-  v    silkFlowtypeName (6871, 938)       17    class & type
  v-  v    silkClassName (6871, 939)         18    class
  v-  v    silkTypeName (6871, 940)          19    type
  v-  v    silkSensorName (6871, 941)        20    sensor
```

4. Template ID 0x9DD3 (40403), for IPv4 records whose protocol is TCP and that do not have the expanded TCP flags fields (initial flags and session flags):

OCTETS	INFORMATION ELEMENT (PEN, ID)	POS	SILK FIELD
0- 7	flowStartMilliseconds (152)	1	sTime
8- 15	flowEndMilliseconds (153)	2	sTime + duration
16- 19	packetDeltaCount (2)	3	packets
20- 23	octetDeltaCount (1)	4	bytes
24- 25	ingressInterface (10)	5	in
26- 27	egressInterface (14)	6	out
28- 29	silkAppLabel (6871, 33)	7	application
30- 31	silkSensorId (6871, 31)	8	sensor
32	silkFlowtypeId (6871, 30)	9	class & type
33	silkTCPState (6871, 32)	10	attributes
34	protocolIdentifier (4)	11	protocol
35	tcpControlBits (6)	12	flags
36- 37	sourceTransportPort (7)	13	sPort
38- 39	destinationTransportPort (11)	14	dPort
40- 43	sourceIPv4Address (8)	15	sIP
44- 47	destinationIPv4Address (12)	16	dIP
48- 51	ipNextHopIPv4Address (15)	17	nhIP
52- v	silkFlowtypeName (6871, 938)	18	class & type
v- v	silkClassName (6871, 939)	19	class
v- v	silkTypeName (6871, 940)	20	type
v- v	silkSensorName (6871, 941)	21	sensor

5. Template ID 0x9DD4 (40404), for IPv4 records whose protocol is TCP and that have have the initial flags and session flags fields:

OCTETS	INFORMATION ELEMENT (PEN, ID)	POS	SILK FIELD
0- 7	flowStartMilliseconds (152)	1	sTime
8- 15	flowEndMilliseconds (153)	2	sTime + duration
16- 19	packetDeltaCount (2)	3	packets
20- 23	octetDeltaCount (1)	4	bytes
24- 25	ingressInterface (10)	5	in
26- 27	egressInterface (14)	6	out
28- 29	silkAppLabel (6871, 33)	7	application
30- 31	silkSensorId (6871, 31)	8	sensor
32	silkFlowtypeId (6871, 30)	9	class & type
33	silkTCPState (6871, 32)	10	attributes
34	protocolIdentifier (4)	11	protocol
35	initialTCPFlags (6871, 14)	12	initialFlags
36	unionTCPFlags (6871, 15)	13	sessionFlags
37	tcpControlBits (6)	14	flags
38- 39	sourceTransportPort (7)	15	sPort
40- 41	destinationTransportPort (11)	16	dPort
42- 45	sourceIPv4Address (8)	17	sIP
46- 49	destinationIPv4Address (12)	18	dIP
50- 53	ipNextHopIPv4Address (15)	19	nhIP
54- v	silkFlowtypeName (6871, 938)	20	class & type

```
 v-  v    silkClassName (6871, 939)         21    class
 v-  v    silkTypeName (6871, 940)          22    type
 v-  v    silkSensorName (6871, 941)        23    sensor
```

6. Template ID 0x9ED0 (40656), for IPv6 records whose protocol is not ICMP, ICMPv6, UDP, SCTP, or TCP:

```
 OCTETS    INFORMATION ELEMENT (PEN, ID)    POS    SILK FIELD
=======    =============================    ===    =============
 0-  7    flowStartMilliseconds (152)        1    sTime
 8- 15    flowEndMilliseconds (153)          2    sTime + duration
16- 19    packetDeltaCount (2)               3    packets
20- 23    octetDeltaCount (1)                4    bytes
24- 25    ingressInterface (10)              5    in
26- 27    egressInterface (14)               6    out
28- 29    silkAppLabel (6871, 33)            7    application
30- 31    silkSensorId (6871, 31)            8    sensor
32        silkFlowtypeId (6871, 30)          9    class & type
33        silkTCPState (6871, 32)           10    attributes
34        protocolIdentifier (4)            11    protocol
35- 50    sourceIPv6Address (27)            12    sIP
51- 66    destinationIPv6Address (28)       13    dIP
67- 82    ipNextHopIPv6Address (62)         14    nhIP
83-  v    silkFlowtypeName (6871, 938)      15    class & type
 v-  v    silkClassName (6871, 939)         16    class
 v-  v    silkTypeName (6871, 940)          17    type
 v-  v    silkSensorName (6871, 941)        18    sensor
```

7. Template ID 0x9ED1 (40657), for ICMPv6 IPv6 records:

```
 OCTETS    INFORMATION ELEMENT (PEN, ID)    POS    SILK FIELD
=======    =============================    ===    =============
 0-  7    flowStartMilliseconds (152)        1    sTime
 8- 15    flowEndMilliseconds (153)          2    sTime + duration
16- 19    packetDeltaCount (2)               3    packets
20- 23    octetDeltaCount (1)                4    bytes
24- 25    ingressInterface (10)              5    in
26- 27    egressInterface (14)               6    out
28- 29    silkAppLabel (6871, 33)            7    application
30- 31    silkSensorId (6871, 31)            8    sensor
32        silkFlowtypeId (6871, 30)          9    class & type
33        silkTCPState (6871, 32)           10    attributes
34        protocolIdentifier (4)            11    protocol
35- 36    icmpTypeCodeIPv6                  12    dPort
37- 52    sourceIPv6Address (27)            13    sIP
53- 68    destinationIPv6Address (28)       14    dIP
69- 84    ipNextHopIPv6Address (62)         15    nhIP
85-  v    silkFlowtypeName (6871, 938)      16    class & type
 v-  v    silkClassName (6871, 939)         17    class
 v-  v    silkTypeName (6871, 940)          18    type
 v-  v    silkSensorName (6871, 941)        19    sensor
```

8. Template ID 0x9ED2 (40658), for IPv6 records whose protocol is UDP or SCTP:

OCTETS	INFORMATION ELEMENT (PEN, ID)	POS	SILK FIELD
0- 7	flowStartMilliseconds (152)	1	sTime
8- 15	flowEndMilliseconds (153)	2	sTime + duration
16- 19	packetDeltaCount (2)	3	packets
20- 23	octetDeltaCount (1)	4	bytes
24- 25	ingressInterface (10)	5	in
26- 27	egressInterface (14)	6	out
28- 29	silkAppLabel (6871, 33)	7	application
30- 31	silkSensorId (6871, 31)	8	sensor
32	silkFlowtypeId (6871, 30)	9	class & type
33	silkTCPState (6871, 32)	10	attributes
34	protocolIdentifier (4)	11	protocol
35- 36	sourceTransportPort (7)	12	sPort
37- 38	destinationTransportPort (11)	13	dPort
39- 54	sourceIPv6Address (27)	14	sIP
55- 70	destinationIPv6Address (28)	15	dIP
71- 86	ipNextHopIPv6Address (62)	16	nhIP
87- v	silkFlowtypeName (6871, 938)	17	class & type
v- v	silkClassName (6871, 939)	18	class
v- v	silkTypeName (6871, 940)	19	type
v- v	silkSensorName (6871, 941)	20	sensor

9. Template ID 0x9ED3 (40659), for IPv6 records whose protocol is TCP and that do not have the expanded TCP flags fields (initial flags and session flags):

OCTETS	INFORMATION ELEMENT (PEN, ID)	POS	SILK FIELD
0- 7	flowStartMilliseconds (152)	1	sTime
8- 15	flowEndMilliseconds (153)	2	sTime + duration
16- 19	packetDeltaCount (2)	3	packets
20- 23	octetDeltaCount (1)	4	bytes
24- 25	ingressInterface (10)	5	in
26- 27	egressInterface (14)	6	out
28- 29	silkAppLabel (6871, 33)	7	application
30- 31	silkSensorId (6871, 31)	8	sensor
32	silkFlowtypeId (6871, 30)	9	class & type
33	silkTCPState (6871, 32)	10	attributes
34	protocolIdentifier (4)	11	protocol
35	tcpControlBits (6)	12	flags
36- 37	sourceTransportPort (7)	13	sPort
38- 39	destinationTransportPort (11)	14	dPort
40- 55	sourceIPv6Address (27)	15	sIP
56- 71	destinationIPv6Address (28)	16	dIP
72- 87	ipNextHopIPv6Address (62)	17	nhIP
88- v	silkFlowtypeName (6871, 938)	18	class & type
v- v	silkClassName (6871, 939)	19	class
v- v	silkTypeName (6871, 940)	20	type
v- v	silkSensorName (6871, 941)	21	sensor

10. Template ID 0x9ED4 (40660), for IPv6 records whose protocol is TCP and that have have the initial flags and session flags fields:

OCTETS	INFORMATION ELEMENT (PEN, ID)	POS	SILK FIELD
0- 7	flowStartMilliseconds (152)	1	sTime
8- 15	flowEndMilliseconds (153)	2	sTime + duration
16- 19	packetDeltaCount (2)	3	packets
20- 23	octetDeltaCount (1)	4	bytes
24- 25	ingressInterface (10)	5	in
26- 27	egressInterface (14)	6	out
28- 29	silkAppLabel (6871, 33)	7	application
30- 31	silkSensorId (6871, 31)	8	sensor
32	silkFlowtypeId (6871, 30)	9	class & type
33	silkTCPState (6871, 32)	10	attributes
34	protocolIdentifier (4)	11	protocol
35	initialTCPFlags (6871, 14)	12	initialFlags
36	unionTCPFlags (6871, 15)	13	sessionFlags
37	tcpControlBits (6)	14	flags
38- 39	sourceTransportPort (7)	15	sPort
40- 41	destinationTransportPort (11)	16	dPort
42- 57	sourceIPv6Address (27)	17	sIP
58- 73	destinationIPv6Address (28)	18	dIP
74- 89	ipNextHopIPv6Address (62)	19	nhIP
90- v	silkFlowtypeName (6871, 938)	20	class & type
v- v	silkClassName (6871, 939)	21	class
v- v	silkTypeName (6871, 940)	22	type
v- v	silkSensorName (6871, 941)	23	sensor

When the **--single-template** switch is provided, **rwipfix2silk** uses a single IPFIX template for all records. That template has ID 0xAFEA (45034) and contains the following information elements:

OCTETS	INFORMATION ELEMENT (PEN, ID)	POS	SILK FIELD
0- 7	flowStartMilliseconds (152)	1	sTime
8- 15	flowEndMilliseconds (153)	2	sTime + duration
16- 31	sourceIPv6Address (27)	3	sIP
32- 47	destinationIPv6Address (28)	4	dIP
48- 51	sourceIPv4Address (8)	5	sIP
52- 55	destinationIPv4Address (12)	6	dIP
56- 57	sourceTransportPort (7)	7	sPort
58- 59	destinationTransportPort (11)	8	dPort
60- 63	ipNextHopIPv4Address (15)	9	nhIP
64- 79	ipNextHopIPv6Address (62)	10	nhIP
80- 83	ingressInterface (10)	11	in
84- 87	egressInterface (14)	12	out
88- 95	packetDeltaCount (2)	13	packets
96-103	octetDeltaCount (1)	14	bytes
104	protocolIdentifier (4)	15	protocol
105	silkFlowtypeId (6871, 30)	16	class & type
106-107	silkSensorId (6871, 31)	17	sensor

```
108         tcpControlBits (6)                18    flags
109         initialTCPFlags (6871, 14)        19    initialFlags
110         unionTCPFlags (6871, 15)          20    sessionFlags
111         silkTCPState (6871, 32)           21    attributes
112-113     silkAppLabel (6871, 33)           22    application
114-119     paddingOctets (210)               23    -
```

Note that the template contains both IPv4 and IPv6 addresses. One set of those addresses contains the IP addresses and the other set contains only zeros. The template never includes elements 938--941.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--ipfix-output=*PATH*

Write the IPFIX records to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output. If *PATH* names an existing file, **rwsilk2ipfix** exits with an error unless the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. If this switch is not given, the output is written to the standard output. Attempting to write the binary output to a terminal causes **rwipfix2silk** to exit with an error.

--no-site-name-elements

Do not export the elements that use the site configuration file (**silk.conf(5)**) to get the names of the flowtype, class, type, and sensor. That is, do not include silkFlowtypeName, silkClassName, silkTypeName, and silkSensorName in the exported templates and records. *Since SiLK 3.20.0.*

--print-statistics

Print, to the standard error, the number of records that were written to the IPFIX output file.

--single-template

Use a single IPFIX template for all records. Using this switch produces output identical to that produced by **rwsilk2ipfix** from SiLK 3.11.0 and earlier. *Since SiLK 3.12.0.*

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwsilk2ipfix** searches for the site configuration file in the locations specified in the FILES section.

--xargs

--xargs=*FILENAME*

Read the names of the input files from *FILENAME* or from the standard input if *FILENAME* is not provided. The input is expected to have one filename per line. **rwsilk2ipfix** opens each named file in turn and reads records from it as if the filenames had been listed on the command line.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line.

To convert the SiLK file *silk.rw* into an IPFIX format and store the results in *ipfix.dat*:

```
$ rwsilk2ipfix --ipfix-output=ipfix.dat silk.rw
```

To view the contents of *ipfix.dat* using the **yafscii(1)** tool (see http://tools.netsa.cert.org/yaf/):

```
$ yafscii --in=ipfix.dat --out=-
```

To view the contents of *ipfix.dat* using the **ipfixDump(1)** tool (see http://tools.netsa.cert.org/yaf/):

```
$ ipfixDump --yaf --in=ipfix.dat --out=-
```

Use the **rwipfix2silk(1)** tool to convert the IPFIX file back into SiLK Flow format:

```
$ rwipfix2silk --silk-output=silk2.rw ipfix.dat
```

ENVIRONMENT

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwsilk2ipfix** may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **rwsilk2ipfix** may use this environment variable. See the FILES section for details.

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

SEE ALSO

rwipfix2silk(1), **rwcut(1)**, **rwflowpack(8)**, **silk.conf(5)**, **silk(7)**, **yaf(1)**, **yafscii(1)**, **ipfixDump(1)**, **applabel(1)**

rwsiteinfo

Print information from the silk.conf site configuration file

SYNOPSIS

```
rwsiteinfo --fields=FIELD[,FIELD...]
      { [--classes=CLASS[,CLASS...]] [--types=TYPE[,TYPE...]]
        | [--flowtypes=CLASS/TYPE[,CLASS/TYPE...]] }
      [--sensors=SENSOR[,SENSOR...]] [--groups=GROUP[,GROUP...]]
      [--data-rootdir=ROOT_DIRECTORY] [--site-config-file=FILENAME]
      [--timestamp-format=FORMAT] [--no-titles]
      [--no-columns] [--column-separator=CHAR]
      [--no-final-delimiter] [{--delimited | --delimited=CHAR}]
      [--list-delimiter=CHAR] [--output-path=PATH]
      [--pager=PAGER_PROG]

rwsiteinfo --help

rwsiteinfo --help-fields

rwsiteinfo --version
```

DESCRIPTION

rwsiteinfo is a utility to print selected information about the classes, types, flowtypes, sensors, and groups (collections of sensors) that are defined in the **silk.conf(5)** site configuration file. The **--fields** switch is required, and its argument is a comma-separated list of field names selecting the fields to be printed. The output from **rwsiteinfo** consists of multiple columns and rows, where each column contains one of the *FIELD*s and where each row has a unique value for one of the *FIELD*s. **rwsiteinfo** prints rows until all possible combinations of fields is exhausted. By default, the information is printed in a columnar, bar-delimited (|-delimited) format.

As of SiLK 3.11.0, **rwsiteinfo** can visit the files in the data repository to report the date of the earliest (oldest) file in the repository, the date of the latest (most recent) file in the repository, and the number of files in the repository. These values are reported individually for each row in the output. **Note:** If your data repository is large, scanning it may take a long time.

The **--classes**, **--types**, **--flowtypes**, **--sensors**, and **--groups** switches allow the user to limit the amount of information printed. (These switches operate similarly to their namesakes on **rwfilter(1)** and **rwfglob(1)**.) If none of these switches are given, **rwsiteinfo** prints information for *all* values defined in the *silk.conf* file. If one or more of these switches is specified, **rwsiteinfo** limits its output to the specified values. The limit is applied even if that field is not listed in **--fields**. For example, specifying **--sensors=S1 --fields=class** tells **rwsiteinfo** to limit the output to classes that have sensor `S1` as a member. To print information about the default class or the default types within a class, use the at-sign (`@`) as the name of the class or type, respectively. The **--flowtypes** switch must be used independently of the **--classes** and **--types** switches.

SiLK 3.21.0 added support for sensor groups to **rwsiteinfo**. Sensor groups are defined in the **silk.conf(5)** file within a **group** block. The **--sensors** switch accepts sensor group names to limit the output to sensors defined with the named groups. The **--groups** switch may be used to limit the output to specified sensor

groups. Using the same argument to **--groups** or **--sensors** produces the same result unless **--fields** includes group and the *silk.conf* file has sensors as members of multiple groups.

As of SiLK 3.20, the **--classes**, **--types**, **--flowtypes**, **--sensors**, and **--groups** switches accept a value in the form @*PATH*, where *PATH* names a file (that is, an `@` character followed by a file or path name). The format of this file is described in the Read Argument Values from a File section below.

As stated above, **rwsiteinfo** prints unique rows given a list of *FIELD*s. As an example, suppose the user entered the command `rwsiteinfo --fields=class,type,sensor`. **rwsiteinfo** will print a row containing the first class defined in the *silk.conf* file, the first type defined for that class, and the first sensor name defined for that class/type pair. On the next row, the class and type will be the same and the second sensor name will be printed. Once all sensors have been printed, **rwsiteinfo** repeats the process for the second type defined for the first class, and so on. Once all information for the first class has been printed, the process would repeat for the next class, until all classes have been printed.

The order of the *FIELD*s determines how **rwsiteinfo** iterates through the possible values. The last *FIELD* will change most rapidly, and the first field will change most slowly. Two invocations of **rwsiteinfo** where the first specifies `--fields=class,sensor` and the second specifies `--fields=sensor,class` produce the same number of rows, and each invocation has an outer and inner iterator. In the first invocation, the outer iterator is over the classes, and the inner iterator is over each sensor defined in that class. In the second invocation, the outer iterator is over the sensors, and the inner is over the classes to which that sensor belongs.

In general, the output will contain some combination of class, type, flowtype, and sensor. For flowtype and sensor, the numeric ID may be printed instead of the name. For class and type, the default values may be printed or they may be identified by a symbol. Most field names support a *FIELD***:list** variant that puts all possible values for that field into a single column. See the description of the **--fields** switch below for details.

Read Argument Values from a File

As of SiLK 3.20, the **--classes**, **--types**, **--flowtypes**, **--sensors**, and **--group** switches accept a value in the form @*PATH*, where `@` is the "at" character (ASCII 0x40) and *PATH* names a file or a path to a file. For example, the following reads the name of types from the file *t.txt* and uses the sensors `S3`, `S7`, and the names and/or IDs read from */tmp/sensor.txt*:

```
rwsiteinfo --type=@t.txt --sensors=S3,@/tmp/sensor.txt,S7
```

Multiple @*PATH* values are allowed within a single argument. If the name of the file is `-`, the names are read from the standard input.

The file must be a text file. Blank lines are ignored as are comments, which begin with the `#` character and continue to the end of the line. Whitespace at the beginning and end of a line is ignored as is whitespace that surrounds commas; all other whitespace within a line is significant.

A file may contain a value on each line and/or multiple values on a line separated by commas and optional whitespace. For example:

```
# Sensor 4
      S4
# The first sensors
S0, S1,S2
S3     # Sensor 3
```

An attempt to use an @*PATH* directive in a file is an error.

When **rwsiteinfo** is parsing the name of a file, it converts the sequences `@,` and `@@` to `,` and `@`, respectively. For example, **--class=@cl@@ss.txt@,v** reads the class from the file *cl@ss.txt,v*. It is an error if any other character follows an embedded `@` (**--flowtypes=@f@il** contains `@i`) or if a single `@` occurs at the end of the name (**--sensor=@errat@**).

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--fields=*FIELD*[,*FIELD*...]

Specify the fields to print as a comma-separated list of names. The names are case-insensitive. The fields will be displayed in the order the names are specified. The **--fields** switch is required, and **rwsiteinfo** will fail when it is not provided.

The list of possible field names is:

class
: the class name, e.g., `all`

type
: the type name, e.g., `inweb`

flowtype
: the flowtype name, e.g., `iw`. The flowtype name is a combination of the class name and type name, and it is used to name files in the SiLK data repository.

id-flowtype
: the integer identifier for the flowtype, e.g., `2`

sensor
: the sensor name, e.g., `S3`

id-sensor
: the integer identifier for the sensor, e.g., `3`

describe-sensor
: the sensor description, when present

group
: the sensor group name *Since SiLK 3.21.0.*

default-class
: the default class name

default-type
: the default type name

mark-defaults
: a two-character wide column that contains a plus '+' on a row that contains the default class and an asterisk '*' on a row that contains a default type

repo-start-date
: the earliest date for a file in the repository that matches the values in this row or empty when no files match *Since SiLK 3.11.0*

repo-end-date
: the latest date for a file in the repository that matches the values in this row or empty when no files match *Since SiLK 3.11.0*

repo-file-count
: the number of files in the repository that match the values in this row or zero when no files match *Since SiLK 3.11.0*

class:list
: instead of printing class names on separate rows, join all the classes in a single row separated using the list-delimiter

type:list
: instead of printing type names on separate rows, join all the types in a single row separated using the list-delimiter

flowtype:list
: instead of printing flowtype names on separate rows, join all the flowtypes in a single row separated using the list-delimiter

id-flowtype:list
: instead of printing flowtype identifiers on separate rows, join all the flowtype identifiers in a single row separated using the list-delimiter

sensor:list
: instead of printing sensor names on separate rows, join all the sensors in a single row separated using the list-delimiter

id-sensor:list
: instead of printing sensor identifiers on separate rows, join all the sensor identifiers in a single row separated using the list-delimiter

group:list
: instead of printing sensor group names on separate rows, join all the group names in a single row separated using the list-delimiter *Since SiLK 3.21.0.*

default-class:list
: equivalent to default-class, but provided for consistency

default-type:list
: instead of printing the default type names on separate rows, join all the default type names in a single row separated using the list-delimiter

--classes=*CLASS*[,*CLASS*,@*PATH*...]

Restrict the output using the class(es) named in the comma-separated list. The default class may be specified by using an at-sign (`@`) as the name of a class. As of SiLK 3.20.0, if an argument is "@*PATH*", **rwsiteinfo** attempts to open the file *PATH* and read the names of classes from it; see Read Argument Values from a File for details.

--types=*TYPE*[,*TYPE*,@*PATH*...]

Restrict the output using the type(s) named in the comma-separated list. The default types for a class may be specified by using an at-sign (`@`) as the name of a type. An argument of "@*PATH*" causes **rwsiteinfo** to read type names from the file *PATH*; see Read Argument Values from a File.

--flowtypes=*CLASS*/*TYPE*[,*CLASS*/*TYPE*,@*PATH*...]

Restrict the output using the class/type pairs named in the comma-separated list, where the class name and type name are separated by a slash (`/`). The keyword `all` may be used for the *CLASS*

and/or *TYPE* to select all classes and/or types. As of SiLK 3.20.0, the arguments may also include "@*PATH*" which causes **rwsiteinfo** to open the file *PATH* and read the class/type pairs from it; see Read Argument Values from a File.

--sensors=*SENSOR*[,*SENSOR*,*SENSOR-GROUP*,@*PATH*...]

Restrict the output to the sensors(s) identified in the comma-separated list of sensor names, sensor IDs (integers), ranges of sensor IDs, sensors added to the specified sensor group names, and names and/or IDs of sensors and sensor groups read from the file *PATH*. Using a sensor group name adds all sensors defined within that group. The ability to use sensor group names was added in SiLK 3.21.0.

--groups=*SENSOR-GROUP*[,*SENSOR-GROUP*,@*PATH*...]

Restrict the output using the sensor groups named in the comma-separated list. Naming a group in either **--groups** or **--sensors** produces the same result unless **--fields** includes `group` and the *silk.conf* puts sensors in multiple groups. If an argument is "@*PATH*", **rwsiteinfo** attempts to open the file *PATH* and read the names of sensor groups from it; see Read Argument Values from a File for details. *Since SiLK-3.21.0.*

--data-rootdir=*ROOT_DIRECTORY*

Use *ROOT_DIRECTORY* as the root of the data repository, which overrides the location given in the SILK_DATA_ROOTDIR environment variable, which in turn overrides the location that was compiled into **rwsiteinfo** (/data). This directory is one of the locations where **rwsiteinfo** attempts to find the *silk.conf* file, and it is the repository that is scanned when the repo-start-date, repo-end-date, or repo-file-count field is specified.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwsiteinfo** searches for the site configuration file in the locations specified in the FILES section.

--timestamp-format=*FORMAT*

Specify the format and/or timezone to use when printing timestamps. When this switch is not specified, the SILK_TIMESTAMP_FORMAT environment variable is checked for a default format and/or timezone. If it is empty or contains invalid values, timestamps are printed in the default format, and the timezone is UTC unless SiLK was compiled with local timezone support. *FORMAT* is a comma-separated list of a format and/or a timezone. The format is one of:

default
: Print the timestamps as *YYYY*/*MM*/*DD*T*hh*:*mm*:*ss*.

iso
: Print the timestamps as *YYYY*-*MM*-*DD* *hh*:*mm*:*ss*.

m/d/y
: Print the timestamps as *MM*/*DD*/*YYYY* *hh*:*mm*:*ss*.

epoch
: Print the timestamps as the number of seconds since 00:00:00 UTC on 1970-01-01.

When a timezone is specified, it is used regardless of the default timezone support compiled into SiLK. The timezone is one of:

utc
: Use Coordinated Universal Time to print timestamps.

local
: Use the TZ environment variable or the local timezone.

--no-titles

Turn off column titles. By default, titles are printed.

--no-columns

Disable fixed-width columnar output.

--column-separator=*C*

Use specified character between columns and after the final column. When this switch is not specified, the default of | is used.

--no-final-delimiter

Do not print the column separator after the final column. Normally a delimiter is printed.

--delimited

--delimited=*C*

Run as if **--no-columns --no-final-delimiter --column-sep**=*C* had been specified. That is, disable fixed-width columnar output; if character *C* is provided, it is used as the delimiter between columns instead of the default |.

--list-delimiter=*C*

Specify the character to use between items that comprise a *FIELD*:**list** column. The default list delimiter is comma `,`.

--output-path=*PATH*

Write the textual output to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output (and bypass the paging program). If *PATH* names an existing file, **rwsiteinfo** exits with an error unless the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. If this switch is not given, the output is either sent to the pager or written to the standard output. *Since SiLK 3.15.0.*

--pager=*PAGER_PROG*

When output is to a terminal, invoke the program *PAGER_PROG* to view the output one screen full at a time. This switch overrides the SILK_PAGER environment variable, which in turn overrides the PAGER variable. If the **--output-path** switch is given or if the value of the pager is determined to be the empty string, no paging is performed and all output is written to the terminal.

--help

Print the available options and exit. Options that add fields can be specified before **--help** so that the new options appear in the output.

--help-fields

Print a description for each field and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

The output from these examples is based on the sample *silk.conf* as distributed for the twoway site (c.f. **packlogic-twoway(3)**).

Displaying the sensors with various options

This displays all known sensors using the default display options:

```
$ rwsiteinfo --fields=sensor
Sensor|
    S0|
    S1|
    S2|
    S3|
    S4|
    S5|
    S6|
    S7|
    S8|
    S9|
   S10|
   S11|
   S12|
   S13|
   S14|
```

The following prints all known sensor names, one name per line:

```
$ rwsiteinfo --fields=sensor --no-titles --delimited
S0
S1
S2
S3
S4
S5
S6
S7
S8
S9
S10
S11
S12
S13
S14
```

The following prints all known sensor names on a single line with the names separated by comma:

```
$ rwsiteinfo --fields=sensor:list --no-titles --delimited
S0,S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,S13,S14
```

This changes the output from the previous example to use a space as the separator:

```
$ rwsiteinfo --fields=sensor:list --no-titles --delimited \
       --list-delimiter=' '
S0 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14
```

The following prints the sensor names for the default class on a single line (since there is a single class, the output is the same as that shown above):

```
$ rwsiteinfo --fields=sensor:list --class=@ --no-titles --delimited
S0,S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,S13,S14
```

This shows the numeric sensor IDs:

```
$ rwsiteinfo --fields=id-sensor:list
                    Sensor-ID:list|
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14|
```

The following prints four columns: (1) the sensor identifier, (2) the sensor name, (3) the list of classes for that sensor, and (4) a description of the sensor. This output mimics the output of the deprecated **mapsid(1)** tool.

```
$ rwsiteinfo --fields=id-sensor,sensor,class:list,describe-sensor
Sensor-ID|Sensor|Class:list|               Sensor-Description|
        0|    S0|       all|         Description for sensor S0|
        1|    S1|       all|                                  |
        2|    S2|       all|Optional description for sensor S2|
        3|    S3|       all|                                  |
        4|    S4|       all|                                  |
        5|    S5|       all|                                  |
        6|    S6|       all|                                  |
        7|    S7|       all|                                  |
        8|    S8|       all|                                  |
        9|    S9|       all|                                  |
       10|   S10|       all|                                  |
       11|   S11|       all|                                  |
       12|   S12|       all|                                  |
       13|   S13|       all|                                  |
       14|   S14|       all|                                  |
```

Displaying classes and types

This prints three columns: the first contains the class, the second contains the type, and the third uses a `+` to mark rows for the default class and a `*` to mark rows for a default type.

```
$ rwsiteinfo --fields=class,type,mark-default
Class|   Type|Defaults|
  all|     in|      +*|
  all|    out|      + |
  all|  inweb|      +*|
  all| outweb|      + |
  all| innull|      + |
  all|outnull|      + |
  all|int2int|      + |
  all|ext2ext|      + |
  all| inicmp|      +*|
  all|outicmp|      + |
  all|  other|      + |
```

The following prints two columns, the first containing a class name and the second the list of default types for that class:

```
$ rwsiteinfo --fields=class,default-type:list
Class|Default-Type:list|
  all|  in,inweb,inicmp|
```

The following prints the default types. (The output contains the default type for each class, but twoway site has a single class.)

```
$ rwsiteinfo --fields=default-type --no-titles --delimited
in
inweb
inicmp
```

This does the same (by limiting the output the default types).

```
$ rwsiteinfo --fields=type --types=@ --no-titles --delimited
in
inweb
inicmp
```

The following prints the class, the sensor, and the type. The number of rows of output (excluding the title) is the product of the number of classes, number of types, and number of sensors.

```
$ rwsiteinfo --fields=class,sensor,type
Class|Sensor|   Type|
  all|    S0|     in|
  all|    S0|    out|
  all|    S0|  inweb|
  all|    S0| outweb|
  all|    S0| innull|
  all|    S0|outnull|
  all|    S0|int2int|
  all|    S0|ext2ext|
  all|    S0| inicmp|
```

```
  all|    S0|outicmp|
  all|    S0|  other|
  all|    S1|     in|
  all|    S1|    out|
...
  all|   S14|outicmp|
  all|   S14|  other|
```

Displaying sensor groups

The **--group** switch was added in SiLK 3.21.0 and limits the output to particular sensor groups. In addition, the **--sensors** switch accepts sensor group names. Given this *silk.conf* file:

```
sensor 1 S1
sensor 2 S2
sensor 3 S3
group G12
  sensors S1 S2
end group
group G13
  sensors S1 S3
end group
...
```

Using **--group=G12** limits the output to that group:

```
$ rwsiteinfo --fields=group,sensor --group=G12
Group|Sensor|
  G12|    S1|
  G12|    S2|
```

If **--sensors=G12** is used, **rwsiteinfo** limits the output to the sensors that group contains (**S1** and **S2**). The result includes group **G13** since that group contains sensor **S1**:

```
$ rwsiteinfo --fields=group,sensor --sensor=G12
Group|Sensor|
  G12|    S1|
  G12|    S2|
  G13|    S1|
```

This occurs when sensors appear in multiple groups and group is one of the output **--fields**.

Displaying information about the flow data repository

The repo-start-date, repo-end-date, and repo-file-count fields print the range of available dates for the files in the repository. The following shows information about files in the repository for the repository as a whole:

```
$ rwsiteinfo --fields=repo-start-date,repo-end-date,repo-file-count
         Start-Date|           End-Date|File-Count|
2009/02/12T00:00:00|2009/02/14T23:00:00|      2880|
```

This breaks down the file information per type:

```
$ rwsiteinfo --fields=type,repo-start-date,repo-end-date,repo-file-count
   Type|         Start-Date|           End-Date|File-Count|
     in|2009/02/12T00:00:00|2009/02/14T23:00:00|       720|
    out|2009/02/12T00:00:00|2009/02/14T23:00:00|       720|
  inweb|2009/02/12T00:00:00|2009/02/14T23:00:00|       720|
 outweb|2009/02/12T00:00:00|2009/02/14T23:00:00|       720|
 innull|                   |                   |         0|
outnull|                   |                   |         0|
int2int|                   |                   |         0|
ext2ext|                   |                   |         0|
 inicmp|                   |                   |         0|
outicmp|                   |                   |         0|
  other|                   |                   |         0|
```

This shows the information for each sensor:

```
$ rwsiteinfo --fields=type,repo-start-date,repo-end-date,repo-file-count
Sensor|         Start-Date|           End-Date|File-Count|
    S0|2009/02/12T00:00:00|2009/02/14T23:00:00|       288|
    S1|2009/02/12T00:00:00|2009/02/14T23:00:00|       288|
    S2|2009/02/12T00:00:00|2009/02/14T23:00:00|       288|
    S3|2009/02/12T00:00:00|2009/02/14T23:00:00|       288|
    S4|2009/02/12T00:00:00|2009/02/14T23:00:00|       288|
    S5|2009/02/12T00:00:00|2009/02/14T23:00:00|       288|
    S6|2009/02/12T00:00:00|2009/02/14T23:00:00|       288|
    S7|2009/02/12T00:00:00|2009/02/14T23:00:00|       288|
    S8|2009/02/12T00:00:00|2009/02/14T23:00:00|       288|
    S9|2009/02/12T00:00:00|2009/02/14T23:00:00|       288|
   S10|                   |                   |         0|
   S11|                   |                   |         0|
   S12|                   |                   |         0|
   S13|                   |                   |         0|
   S14|                   |                   |         0|
```

Restricting the output by reading names from a file

Suppose the file *sensors.txt* exits in the current directory and has the content shown here:

```
$ cat sensors.txt
# Sensor 4
      S4
# The first sensors
S0,S1,S2
S3     # Sensor 3
```

When using SiLK 3.20.0 or later, the file can be used to restrict which sensors **rwsiteinfo** displays:

```
$ rwsiteinfo --fields=class,sensor  --sensor=@/tmp/foobart
```

```
Class|Sensor|
  all|    S0|
  all|    S1|
  all|    S2|
  all|    S3|
  all|    S4|
```

An example that shows use of the `@` escapes when parsing the name of a file (file *weird,n@me* becomes `weird@,n@@me`), and the error generated when attempting to use @*PATH* within a file:

```
$ cat weird,n@me
S1
@another-file

$ rwsiteinfo --fields=sensor --sensor=@weird@,n@@me
rwsiteinfo: Error parsing sensors '@weird@,n@@me': Bad
    token '@another-file' found in file 'weird,n@me': May not
    recursively use @FILE construct within a file
```

ENVIRONMENT

SILK_TIMESTAMP_FORMAT

This environment variable is used as the value for **--timestamp-format** when that switch is not provided. *Since SiLK 3.11.0.*

SILK_PAGER

When set to a non-empty string, **rwsiteinfo** automatically invokes this program to display its output a screen at a time. If set to an empty string, **rwsiteinfo** does not automatically page its output.

PAGER

When set and SILK_PAGER is not set, **rwsiteinfo** automatically invokes this program to display its output a screen at a time.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwsiteinfo** may use this environment variable when searching for the SiLK site configuration file. In addition, **rwsiteinfo** visits all the files in this directory when the repo-start-date, repo-end-date, or repo-file-count fields are specified in the **--fields** switch.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files and plug-ins, **rwsiteinfo** may use this environment variable. See the FILES section for details.

TZ

When the argument to the **--timestamp-format** switch includes `local` or when a SiLK installation is built to use the local timezone, the value of the TZ environment variable determines the timezone in which **rwsiteinfo** displays timestamps. (If both of those are false, the TZ environment variable is ignored.) If the TZ environment variable is not set, the machine's default timezone is used. Setting TZ to the empty string or 0 causes timestamps to be displayed in UTC. For system information on the TZ variable, see **tzset(3)** or **environ(7)**. (To determine if SiLK was built with support for the local timezone, check the `Timezone support` value in the output of **rwsiteinfo --version**.)

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided. The location of the SILK_DATA_ROOTDIR may be specified using the **--root-directory** switch.

${SILK_DATA_ROOTDIR}/

/data/

Locations for the root directory of the data repository when the **--data-rootdir** switch is not specified.

NOTES

rwsiteinfo added support for sensor groups in SiLK 3.21.0. For information on sensor groups, see the documentation for the **group** block command in the **silk.conf(5)** manual page.

The ability to read classes, types, sensors, and flowtypes from files via the @*PATH* directive was added in SiLK 3.20.0.

The **--output-path** switch was added in SiLK 3.15.0.

The repo-start-date, repo-end-date, and repo-file-count fields were added in SiLK 3.11.0.

rwsiteinfo was added in SiLK 3.0.

rwsiteinfo duplicates the functionality found in **mapsid(1)**. **mapsid** is deprecated, and it will be removed in the SiLK 4.0 release. Examples of using **rwsiteinfo** in place of **mapsid** are provided in the latter's manual page.

SEE ALSO

silk.conf(5), **mapsid(1)**, **rwfilter(1)**, **rwfglob(1)**, **packlogic-twoway(3)**, **silk(7)**, **tzset(3)**, **environ(7)**

rwsort

Sort SiLK Flow records on one or more fields

SYNOPSIS

```
rwsort --fields=KEY [--presorted-input] [--reverse]
      [--temp-directory=DIR_PATH] [--sort-buffer-size=SIZE]
      [--note-add=TEXT] [--note-file-add=FILE]
      [--compression-method=COMP_METHOD] [--print-filenames]
      [--output-path=PATH] [--site-config-file=FILENAME]
      [--plugin=PLUGIN [--plugin=PLUGIN ...]]
      [--python-file=PATH [--python-file=PATH ...]]
      [--pmap-file=MAPNAME:PATH [--pmap-file=MAPNAME:PATH ...]]
      {[--input-pipe=PATH] | [--xargs]|[--xargs=FILE] | [FILES...]}

rwsort [--pmap-file=MAPNAME:PATH [--pmap-file=MAPNAME:PATH ...]]
      [--plugin=PLUGIN ...] [--python-file=PATH ...] --help

rwsort [--pmap-file=MAPNAME:PATH [--pmap-file=MAPNAME:PATH ...]]
      [--plugin=PLUGIN ...] [--python-file=PATH ...] --help-fields

rwsort --version
```

DESCRIPTION

rwsort reads SiLK Flow records, sorts the records by the field(s) listed in the **--fields** switch, and writes the records to the **--output-path** or to the standard output if it is not connected to a terminal. The output from **rwsort** is binary SiLK Flow records; the output must be passed into another tool for human-readable output.

Sorting records is an expensive operation, and it should only be used when necessary. The tools that bin flow records (**rwcount(1)**, **rwuniq(1)**, **rwstats(1)**, etc) do not require sorted data.

rwsort reads SiLK Flow records from the files named on the command line or from the standard input when no file names are specified and neither **--xargs** nor **--input-pipe** is present. To read the standard input in addition to the named files, use `-` or `stdin` as a file name. If an input file name ends in `.gz`, the file is uncompressed as it is read. When the **--xargs** switch is provided, **rwsort** reads the names of the files to process from the named text file or from the standard input if no file name argument is provided to the switch. The input to **--xargs** must contain one file name per line. The **--input-pipe** switch is deprecated and it is provided for legacy reasons; its use is not required since **rwsort** will automatically read form the standard input. The **--input-pipe** switch will be removed in the SiLK 4.0 release.

The amount of fast memory used by **rwsort** will increase until it reaches a maximum near 2GB. (Use the **--sort-buffer-size** switch to change this upper limit on the buffer size.) If more records are read than will fit into memory, the in-core records are sorted and temporarily stored on disk as described by the **--temp-directory** switch. When all records have been read, the on-disk files are merged and the sorted records written to the output.

By default, the temporary files are stored in the */tmp* directory. Because these temporary files will be large, it is strongly recommended that */tmp not* be used as the temporary directory. To modify the tempo-

rary directory used by **rwsort**, provide the **--temp-directory** switch, set the SILK_TMPDIR environment variable, or set the TMPDIR environment variable.

To merge previously sorted SiLK data files into a sorted stream, run **rwsort** with the **--presorted-input** switch. **rwsort** will merge-sort all the input files, reducing it's memory requirements considerably. It is the user's responsibility to ensure that all the input files have been sorted with the same **--fields** value (and **--reverse** if applicable). **rwsort** may still require use of a temporary directory while merging the files (for example, if **rwsort** does not have enough available file handles to open all the input files at once).

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

The **--fields** switch is required. **rwsort** will fail when it is not provided.

--fields=*KEY*

KEY contains the list of flow attributes (a.k.a. fields or columns) that make up the key by which flows are sorted. The fields are in listed in order from primary sort key, secondary key, etc. Each field may be specified once only. *KEY* is a comma separated list of field-names, field-integers, and ranges of field-integers; a range is specified by separating the start and end of the range with a hyphen (**-**). Field-names are case insensitive. Example:

```
--fields=stime,10,1-5
```

There is no default value for the **--fields** switch; the switch must be specified.

The complete list of built-in fields that the SiLK tool suite supports follows, though note that not all fields are present in all SiLK file formats; when a field is not present, its value is 0.

sIP,1
: source IP address

dIP,2
: destination IP address

sPort,3
: source port for TCP and UDP, or equivalent

dPort,4
: destination port for TCP and UDP, or equivalent. See note at `iType`.

protocol,5
: IP protocol

packets,pkts,6
: packet count

bytes,7
: byte count

flags,8
: bit-wise OR of TCP flags over all packets

sTime,9,sTime+msec,22
: starting time of flow (milliseconds resolution)

duration,10,dur+msec,24
: duration of flow (milliseconds resolution)

eTime,11,eTime+msec,23
: end time of flow (milliseconds resolution)

sensor,12
: name or ID of sensor where flow was collected

class,20,type,21
: integer value of the class/type pair assigned to the flow by **rwflowpack(8)**

iType
: the ICMP type value for ICMP or ICMPv6 flows and zero for non-ICMP flows. Internally, SiLK stores the ICMP type and code in the `dPort` field, so there is no need have both `dPort` and `iType` or `iCode` in the sort key. This field was introduced in SiLK 3.8.1.

iCode
: the ICMP code value for ICMP or ICMPv6 flows and zero for non-ICMP flows. See note at `iType`.

icmpTypeCode,25
: equivalent to `iType`,`iCode`. This field may not be mixed with `iType` or `iCode`, and this field is deprecated as of SiLK 3.8.1. Prior to SiLK 3.8.1, specifying the `icmpTypeCode` field was equivalent to specifying the `dPort` field.

Many SiLK file formats do not store the following fields and their values will always be 0; they are listed here for completeness:

in,13
: router SNMP input interface or vlanId if packing tools were configured to capture it (see **sensor.conf(5)**)

out,14
: router SNMP output interface or postVlanId

nhIP,15
: router next hop IP

SiLK can store flows generated by enhanced collection software that provides more information than NetFlow v5. These flows may support some or all of these additional fields; for flows without this additional information, the field's value is always 0.

initialFlags,26
: TCP flags on first packet in the flow

sessionFlags,27
: bit-wise OR of TCP flags over all packets except the first in the flow

attributes,28
: flow attributes set by the flow generator:

  `S`
  : all the packets in this flow record are exactly the same size

  `F`
  : flow generator saw additional packets in this flow following a packet with a FIN flag (excluding ACK packets)

`T`
: flow generator prematurely created a record for a long-running connection due to a timeout. (When the flow generator **yaf(1)** is run with the **--silk** switch, it will prematurely create a flow and mark it with `T` if the byte count of the flow cannot be stored in a 32-bit value.)

`C`
: flow generator created this flow as a continuation of long-running connection, where the previous flow for this connection met a timeout (or a byte threshold in the case of **yaf**).

Consider a long-running ssh session that exceeds the flow generator's *active* timeout. (This is the active timeout since the flow generator creates a flow for a connection that still has activity). The flow generator will create multiple flow records for this ssh session, each spanning some portion of the total session. The first flow record will be marked with a `T` indicating that it hit the timeout. The second through next-to-last records will be marked with `TC` indicating that this flow both timed out and is a continuation of a flow that timed out. The final flow will be marked with a `C`, indicating that it was created as a continuation of an active flow.

application,29
: guess as to the content of the flow. Some software that generates flow records from packet data, such as **yaf**, will inspect the contents of the packets that make up a flow and use traffic signatures to label the content of the flow. SiLK calls this label the *application*; **yaf** refers to it as the *appLabel*. The application is the port number that is traditionally used for that type of traffic (see the */etc/services* file on most UNIX systems). For example, traffic that the flow generator recognizes as FTP will have a value of 21, even if that traffic is being routed through the standard HTTP/web port (80).

The following fields provide a way to label the IPs or ports on a record. These fields require external files to provide the mapping from the IP or port to the label:

sType,16
: categorize the source IP address as `non-routable`, `internal`, or `external` and sort based on the category. Uses the mapping file specified by the SILK_ADDRESS_TYPES environment variable, or the *address_types.pmap* mapping file, as described in **addrtype(3)**.

dType,17
: as **sType** for the destination IP address

scc,18
: the country code of the source IP address. Uses the mapping file specified by the SILK_COUNTRY_CODES environment variable, or the *country_codes.pmap* mapping file, as described in **ccfilter(3)**.

dcc,19
: as **scc** for the destination IP

src-*map-name*
: label contained in the prefix map file associated with *map-name*. If the prefix map is for IP addresses, the label is that associated with the source IP address. If the prefix map is for protocol/port pairs, the label is that associated with the protocol and source port. See also the description of the **--pmap-file** switch below and the **pmapfilter(3)** manual page.

dst-*map-name*
: as **src-*map-name*** for the destination IP address or the protocol and destination port.

sval
: as **src-*map-name*** when no map-name is associated with the prefix map file

dval
: as **dst-*map-name*** when no map-name is associated with the prefix map file

Finally, the list of built-in fields may be augmented by the run-time loading of PySiLK code or plug-ins written in C (also called shared object files or dynamic libraries), as described by the **--python-file** and **--plugin** switches.

--presorted-input

Instruct **rwsort** to merge-sort the input files; that is, **rwsort** assumes the input files have been previously sorted using the same values for the **--fields** and **--reverse** switches as was given for this invocation. This switch can greatly reduce **rwsort**'s memory requirements as a large buffer is not required for sorting the records. If the input files were created with **rwsort**, you can run **rwfileinfo(1)** on the files to see the **rwsort** invocation that created them.

--reverse

Cause **rwsort** to reverse the sort order, causing larger values to occur in the output before smaller values. Normally smaller values appear before larger values.

--plugin=*PLUGIN*

Augment the list of fields by using run-time loading of the plug-in (shared object) whose path is *PLUGIN*. The switch may be repeated to load multiple plug-ins. The creation of plug-ins is described in the **silk-plugin(3)** manual page. When *PLUGIN* does not contain a slash (`/`), **rwsort** will attempt to find a file named *PLUGIN* in the directories listed in the FILES section. If **rwsort** finds the file, it uses that path. If *PLUGIN* contains a slash or if **rwsort** does not find the file, **rwsort** relies on your operating system's **dlopen(3)** call to find the file. When the SILK_PLUGIN_DEBUG environment variable is non-empty, **rwsort** prints status messages to the standard error as it attempts to find and open each of its plug-ins.

--temp-directory=*DIR_PATH*

Specify the name of the directory in which to store data files temporarily when more records have been read that will fit into RAM. This switch overrides the directory specified in the SILK_TMPDIR environment variable, which overrides the directory specified in the TMPDIR variable, which overrides the default, */tmp*.

--sort-buffer-size=*SIZE*

Set the maximum size of the buffer used for sorting the records, in bytes. A larger buffer means fewer temporary files need to be created, reducing the I/O wait times. When this switch is not specified, the default maximum for this buffer is near 2GB. The *SIZE* may be given as an ordinary integer, or as a real number followed by a suffix `K`, `M` or `G`, which represents the numerical value multiplied by 1,024 (kilo), 1,048,576 (mega), and 1,073,741,824 (giga), respectively. For example, 1.5K represents 1,536 bytes, or one and one-half kilobytes. (This value does **not** represent the absolute maximum amount of RAM that **rwsort** will allocate, since additional buffers will be allocated for reading the input and writing the output.) The sort buffer is not used when the **--presorted-input** switch is specified.

--note-add=*TEXT*

Add the specified *TEXT* to the header of the output file as an annotation. This switch may be repeated to add multiple annotations to a file. To view the annotations, use the **rwfileinfo(1)** tool.

--note-file-add=*FILENAME*

Open *FILENAME* and add the contents of that file to the header of the output file as an annotation. This switch may be repeated to add multiple annotations. Currently the application makes no effort to ensure that *FILENAME* contains text; be careful that you do not attempt to add a SiLK data file as an annotation.

--compression-method=*COMP_METHOD*

Specify the compression library to use when writing output files. If this switch is not given, the value in the SILK_COMPRESSION_METHOD environment variable is used if the value names an available compression method. When no compression method is specified, output to the standard output or to named pipes is not compressed, and output to files is compressed using the default chosen when SiLK was compiled. The valid values for *COMP_METHOD* are determined by which external libraries were found when SiLK was compiled. To see the available compression methods and the default method, use the **--help** or **--version** switch. SiLK can support the following *COMP_METHOD* values when the required libraries are available.

none

Do not compress the output using an external library.

zlib

Use the **zlib(3)** library for compressing the output, and always compress the output regardless of the destination. Using zlib produces the smallest output files at the cost of speed.

lzo1x

Use the *lzo1x* algorithm from the LZO real time compression library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead.

snappy

Use the *snappy* library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead. *Since SiLK 3.13.0.*

best

Use lzo1x if available, otherwise use snappy if available, otherwise use zlib if available. Only compress the output when writing to a file.

--print-filenames

Print to the standard error the names of input files as they are opened.

--output-path=*PATH*

Write the binary SiLK Flow records to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output. If *PATH* names an existing file, **rwsort** exits with an error unless the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. If this switch is not given, the output is written to the standard output. Attempting to write the binary output to a terminal causes **rwsort** to exit with an error.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwsort** searches for the site configuration file in the locations specified in the FILES section.

--input-pipe=*PATH*

Read the SiLK Flow records to be sorted from the named pipe at *PATH*. If *PATH* is `stdin` or `-`, records are read from the standard input. Use of this switch is not required, since **rwsort** will automatically read data from the standard input when no file names are specified on the command line. This switch is deprecated and will be removed in the SiLK 4.0 release.

--xargs

--xargs=*FILENAME*

Read the names of the input files from *FILENAME* or from the standard input if *FILENAME* is not provided. The input is expected to have one filename per line. **rwsort** opens each named file in turn and reads records from it as if the filenames had been listed on the command line.

--help

Print the available options and exit. Specifying switches that add new fields or additional switches before **--help** will allow the output to include descriptions of those fields or switches.

--help-fields

Print the description and alias(es) of each field and exit. Specifying switches that add new fields before **--help-fields** will allow the output to include descriptions of those fields.

--version

Print the version number and information about how SiLK was configured, then exit the application.

--pmap-file=*PATH*

--pmap-file=*MAPNAME*:*PATH*

Load the prefix map file located at *PATH* and create fields named src-*map-name* and dst-*map-name* where *map-name* is either the *MAPNAME* part of the argument or the map-name specified when the file was created (see **rwpmapbuild(1)**). If no map-name is available, **rwsort** names the fields `sval` and `dval`. Specify *PATH* as `-` or `stdin` to read from the standard input. The switch may be repeated to load multiple prefix map files, but each prefix map must use a unique map-name. The **--pmap-file** switch(es) must precede the **--fields** switch. See also **pmapfilter(3)**.

--python-file=*PATH*

When the SiLK Python plug-in is used, **rwsort** reads the Python code from the file *PATH* to define additional fields that can be used as part of the sort key. This file should call **register_field()** for each field it wishes to define. For details and examples, see the **silkpython(3)** and **pysilk(3)** manual pages.

LIMITATIONS

When the temporary files and the final output are stored on the same file volume, **rwsort** will require approximately twice as much free disk space as the size of data to be sorted.

When the temporary files and the final output are on different volumes, **rwsort** will require between 1 and 1.5 times as much free space on the temporary volume as the size of the data to be sorted.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line.

To sort the records in *infile.rw* based primarily on destination port and secondarily on source IP and write the binary output to *outfile.rw*, run:

```
$ rwsort --fields=dport,sip --output-path=outfile.rw infile.rw
```

The **silkpython(3)** manual page provides examples that use PySiLK to create arbitrary fields to use as part of the key for **rwsort**.

ENVIRONMENT

SILK_TMPDIR

When set and **--temp-directory** is not specified, **rwsort** writes the temporary files it creates to this directory. SILK_TMPDIR overrides the value of TMPDIR.

TMPDIR

When set and SILK_TMPDIR is not set, **rwsort** writes the temporary files it creates to this directory.

PYTHONPATH

This environment variable is used by Python to locate modules. When **--python-file** is specified, **rwsort** must load the Python files that comprise the PySiLK package, such as *silk/__init__.py*. If this *silk/* directory is located outside Python's normal search path (for example, in the SiLK installation tree), it may be necessary to set or modify the PYTHONPATH environment variable to include the parent directory of *silk/* so that Python can find the PySiLK module.

SILK_PYTHON_TRACEBACK

When set, Python plug-ins will output traceback information on Python errors to the standard error.

SILK_COUNTRY_CODES

This environment variable allows the user to specify the country code mapping file that **rwsort** uses when computing the scc and dcc fields. The value may be a complete path or a file relative to the SILK_PATH. See the FILES section for standard locations of this file.

SILK_ADDRESS_TYPES

This environment variable allows the user to specify the address type mapping file that **rwsort** uses when computing the sType and dType fields. The value may be a complete path or a file relative to the SILK_PATH. See the FILES section for standard locations of this file.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_COMPRESSION_METHOD

This environment variable is used as the value for **--compression-method** when that switch is not provided. *Since SiLK 3.13.0.*

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwsort** may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files and plug-ins, **rwsort** may use this environment variable. See the FILES section for details.

SILK_PLUGIN_DEBUG

When set to 1, **rwsort** prints status messages to the standard error as it attempts to find and open each of its plug-ins. In addition, when an attempt to register a field fails, the application prints a message specifying the additional function(s) that must be defined to register the field in the application. Be aware that the output can be rather verbose.

SILK_TEMPFILE_DEBUG

When set to 1, **rwsort** prints debugging messages to the standard error as it creates, re-opens, and removes temporary files.

FILES

${SILK_ADDRESS_TYPES}

${SILK_PATH}/share/silk/address_types.pmap

${SILK_PATH}/share/address_types.pmap

/usr/local/share/silk/address_types.pmap

/usr/local/share/address_types.pmap

Possible locations for the address types mapping file required by the sType and dType fields.

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

${SILK_COUNTRY_CODES}

${SILK_PATH}/share/silk/country_codes.pmap

${SILK_PATH}/share/country_codes.pmap

/usr/local/share/silk/country_codes.pmap

/usr/local/share/country_codes.pmap

Possible locations for the country code mapping file required by the scc and dcc fields.

${SILK_PATH}/lib64/silk/

${SILK_PATH}/lib64/

${SILK_PATH}/lib/silk/

${SILK_PATH}/lib/

/usr/local/lib64/silk/

/usr/local/lib64/

/usr/local/lib/silk/

/usr/local/lib/

Directories that **rwsort** checks when attempting to load a plug-in.

${SILK_TMPDIR}/

${TMPDIR}/

/tmp/

Directory in which to create temporary files.

SEE ALSO

rwcount(1), **rwcut(1)**, **rwfileinfo(1)**, **rwstats(1)**, **rwuniq(1)**, **rwpmapbuild(1)**, **addrtype(3)**, **cc-filter(3)**, **pmapfilter(3)**, **pysilk(3)**, **silkpython(3)**, **silk-plugin(3)**, **sensor.conf(5)**, **rwflowpack(8)**, **silk(7)**, **yaf(1)**, **dlopen(3)**, **zlib(3)**

NOTES

If an output path is not specified, **rwsort** will write to the standard output unless it is connected to a terminal, in which case an error is printed and **rwsort** exits.

If an input pipe or a set of input files are not specified, **rwsort** will read records from the standard input unless it is connected to a terminal, in which case an error is printed and **rwsort** exits.

Note that **rwsort** produces binary output. Use **rwcut(1)** to view the records.

Do not spend the resources to sort the data if you are going to be passing it to an aggregation tool like **rwtotal** or **rwaddrcount**, which have their own internal data structures that will ignore the sorted data.

Both **rwuniq(1)** and **rwstats(1)** can take advantage of previously sorted data, but you must explicitly inform them that the input is sorted by providing the **--presorted-input** switch.

rwsplit

Divide a SiLK file into a (sampled) collection of subfiles

SYNOPSIS

```
rwsplit --basename=BASENAME
      { --ip-limit=LIMIT | --flow-limit=LIMIT
        | --packet-limit=LIMIT | --byte-limit=LIMIT }
      [--seed=NUMBER] [--sample-ratio=SAMPLE_RATIO]
      [--file-ratio=FILE_RATIO] [--max-outputs=MAX_OUTPUTS]
      [--note-add=TEXT] [--note-file-add=FILE]
      [--compression-method=COMP_METHOD]
      [--print-filenames] [--site-config-file=FILENAME]
      [--xargs[=FILE] | FILE [FILES...]]


rwsplit --help


rwsplit --version
```

DESCRIPTION

rwsplit reads SiLK Flow records from the standard input or from files named on the command line and writes the flows into a set of subfiles based on the splitting criterion. In its simplest form, **rwsplit** *partitions* the file, meaning that each input flow will appear in one (and only one) of the subfiles.

In addition to splitting the file, **rwsplit** can generate files containing sample flows. Sampling is specified by using the **--sample-ratio** and **--file-ratio** switches.

rwsplit reads SiLK Flow records from the files named on the command line or from the standard input when no file names are specified and **--xargs** is not present. To read the standard input in addition to the named files, use `-` or `stdin` as a file name. If an input file name ends in `.gz`, the file is uncompressed as it is read. When the **--xargs** switch is provided, **rwsplit** reads the names of the files to process from the named text file or from the standard input if no file name argument is provided to the switch. The input to **--xargs** must contain one file name per line.

If you wish to use the size of the output files as the splitting criterion, use the **--flow-limit** switch. The paramater to this switch should be the size of the desired output files divided by the record size. The record size can be determined by **rwfileinfo(1)**. When the output files are compressed (see the description of **--compression-method** below), you should assume about a 50% compression ratio.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

The splitting criterion is defined using one of the limit specifiers; one and only one must be specified. They are:

--ip-limit=*LIMIT*

Close the current subfile and begin a new subfile when the count of unique source and destination IPs in the current subfile meets or exceeds *LIMIT*. The next-hop-IP does not count toward *LIMIT*.

--flow-limit=*LIMIT*

Close the current subfile and begin a new subfile when the number of SiLK Flow records in the current subfile meets *LIMIT*.

--packet-limit=*LIMIT*

Close the current subfile and begin a new subfile when the sum of the packet counts across all SiLK Flow records in the current subfile meets or exceeds *LIMIT*.

--byte-limit=*LIMIT*

Close the current subfile and begin a new subfile when the sum of the byte counts across all SiLK Flow records in the current subfile meets or exceeds *LIMIT*. This switch does not specify the size of the subfiles.

The other switches are:

--basename=*BASENAME*

Specifies the basename of the output files; this switch is required. The flows are written sequentially to a set of subfiles whose names follow the format *BASENAME.ORDER.rwf*, where *ORDER* is an 8-digit zero-formatted sequence number (i.e., 00000000, 00000001, and so on). The sequence number will begin at zero and increase by one for every file written, unless **--file-ratio** is specified,

--seed=*NUMBER*

Use *NUMBER* to seed the pseudo-random number generator for the **--sample-ratio** or **--file-ratio** switch. This can be used to put the random number generator into a known state, which is useful for testing.

--sample-ratio=*SAMPLE_RATIO*

Writes one flow record, chosen at random, from every *SAMPLE_RATIO* flows that are read.

--file-ratio=*FILE_RATIO*

Picks one subfile, chosen from random, out of every *FILE_RATIO* names generated, for writing to disk.

--max-outputs=*NUMBER*

Limits the number of files that are written to disk to *NUMBER*.

--note-add=*TEXT*

Add the specified *TEXT* to the header of the output file as an annotation. This switch may be repeated to add multiple annotations to a file. To view the annotations, use the **rwfileinfo(1)** tool.

--note-file-add=*FILENAME*

Open *FILENAME* and add the contents of that file to the header of the output file as an annotation. This switch may be repeated to add multiple annotations. Currently the application makes no effort to ensure that *FILENAME* contains text; be careful that you do not attempt to add a SiLK data file as an annotation.

--compression-method=*COMP_METHOD*

Specify the compression library to use when writing output files. If this switch is not given, the value in the SILK_COMPRESSION_METHOD environment variable is used if the value names an available compression method. When no compression method is specified, the output files are compressed using the default chosen when SiLK was compiled. The valid values for *COMP_METHOD* are determined by which external libraries were found when SiLK was compiled. To see the available compression methods and the default method, use the **--help** or **--version** switch. SiLK can support the following *COMP_METHOD* values when the required libraries are available.

none

Do not compress the output using an external library.

zlib

Use the **zlib(3)** library for compressing the output. Using zlib produces the smallest output files at the cost of speed.

lzo1x

Use the *lzo1x* algorithm from the LZO real time compression library for compression. This compression provides good compression with less memory and CPU overhead.

snappy

Use the *snappy* library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead. *Since SiLK 3.13.0.*

best

Use lzo1x if available, otherwise use snappy if available, otherwise use zlib if available.

--print-filenames

Print to the standard error the names of input files as they are opened.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwsplit** searches for the site configuration file in the locations specified in the FILES section.

--xargs

--xargs=*FILENAME*

Read the names of the input files from *FILENAME* or from the standard input if *FILENAME* is not provided. The input is expected to have one filename per line. **rwsplit** opens each named file in turn and reads records from it as if the filenames had been listed on the command line.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

Assume a source file *source.rwf*; to split that file into files that each contain about 100 unique IP addresses:

```
$ rwsplit --basename=result --ip-limit=100 source.rwf
```

To split *source.rwf* into files that each contain 100 flows:

```
$ rwsplit --basename=result --flow-limit=100 source.rwf
```

The following causes **rwsplit** to sample 1 out of every 10 records from *source.rwf*; i.e., **rwsplit** will read 1000 flow records to produce each subfile:

```
$ rwsplit --basename=result --flow-limit=100 --sample-ratio=10 source.rwf
```

When **--file-ratio** is specified, the file names are generated as usual (e.g., base-00000000, base-00000001, ...); however, one of these names will be chosen randomly from each set of **--file-ratio** candidates, and only that file will be written to disk.

```
$ rwsplit --basename=result --flow-limit=100 --file-ratio=5 source.rwf
$ ls
result-00000002.rwf
result-00000008.rwf
result-00000013.rwf
result-00000016.rwf
```

LIMITATIONS

rwsplit can take *exactly* 1 partitioning switch per invocation.

Partitioning is not exact, **rwsplit** keeps appending flow records a file until it meets or exceeds the specified *LIMIT*. For example, if you specify **--ip-limit**=100, then **rwsplit** will fill up the file until it has 100 IP addresses in it; if the file has 99 addresses and a new record with 2 previously unseen addresses is received, **rwsplit** will put this in the current file, resulting in a 101-address file. Similarly, if you specify **--byte-limit**=2000, and **rwsplit** receives a 10kb flow record, that flow record will be placed in the current subfile.

The switches **--sample-ratio**, **--file-ratio**, and **--max-outputs** are processed in that order. So, when you specify

```
$ rwsplit --sample-ratio=10 --ip-limit=100    \
       --file-ratio=10 --max-outputs=20
```

rwsplit will pick 1 out of every 10 flow records, write that to a file until it has 100 IP's per file, pick 1 out of every 10 files to write, and write up to 20 files. If there are 1000 records, each with 2 unique IPs in them, then **rwsplit** will write **at most** 1 file (it will write 200 unique IP addresses, but it may not pick one of the files from the set to write).

ENVIRONMENT

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_COMPRESSION_METHOD

This environment variable is used as the value for **--compression-method** when that switch is not provided. *Since SiLK 3.13.0.*

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwsplit** may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **rwsplit** may use this environment variable. See the FILES section for details.

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

SEE ALSO

rwfileinfo(1), **silk(7)**, **zlib(3)**

rwstats

Print top-N or bottom-N lists or summarize data by protocol

SYNOPSIS

```
rwstats --fields=KEY [--values=VALUES]
      {--count=N | --threshold=N | --percentage=N}
      [{--top | --bottom}] [--presorted-input] [--no-percents]
      [--ipv6-policy={ignore,asv4,mix,force,only}]
      [{--bin-time=SECONDS | --bin-time}]
      [--timestamp-format=FORMAT] [--epoch-time]
      [--ip-format=FORMAT] [--integer-ips] [--zero-pad-ips]
      [--integer-sensors] [--integer-tcp-flags]
      [--no-titles] [--no-columns] [--column-separator=CHAR]
      [--no-final-delimiter] [{--delimited | --delimited=CHAR}]
      [--print-filenames] [--copy-input=PATH] [--output-path=PATH]
      [--pager=PAGER_PROG] [--temp-directory=DIR_PATH]
      [{--legacy-timestamps | --legacy-timestamps={1,0}}]
      [--site-config-file=FILENAME]
      [--plugin=PLUGIN [--plugin=PLUGIN ...]]
      [--python-file=PATH [--python-file=PATH ...]]
      [--pmap-file=MAPNAME:PATH [--pmap-file=MAPNAME:PATH ...]]
      [--pmap-column-width=NUM]
      {[--xargs] | [--xargs=FILENAME] | [FILE [FILE ...]]}

rwstats {--overall-stats | --detail-proto-stats=PROTO[,PROTO]}
      [--no-titles] [--no-columns] [--column-separator=CHAR]
      [--no-final-delimiter] [{--delimited | --delimited=CHAR}]
      [--print-filenames] [--copy-input=PATH] [--output-path=PATH]
      [--pager=PAGER_PROG]
      {[--xargs] | [--xargs=FILENAME] | [FILE [FILE ...]]}

rwstats [--pmap-file=MAPNAME:PATH [--pmap-file=MAPNAME:PATH ...]]
      [--plugin=PLUGIN ...] [--python-file=PATH ...] --help

rwstats [--pmap-file=MAPNAME:PATH [--pmap-file=MAPNAME:PATH ...]]
      [--plugin=PLUGIN ...] [--python-file=PATH ...] --help-fields

rwstats --legacy-help

rwstats --version
```

DESCRIPTION

rwstats has two modes of operation: it can compute a Top-N or Bottom-N list, or it can summarize data for a list of protocols.

In either mode, **rwstats** reads SiLK Flow records from the files named on the command line or from the standard input when no file names are specified and **--xargs** is not present. To read the standard input

in addition to the named files, use `-` or `stdin` as a file name. If an input file name ends in `.gz`, the file is uncompressed as it is read. When the **--xargs** switch is provided, **rwstats** reads the names of the files to process from the named text file or from the standard input if no file name argument is provided to the switch. The input to **--xargs** must contain one file name per line.

Top-N Description

rwstats reads SiLK Flow records and groups them by a key composed of user-specified attributes of the flows. For each group (or bin), a collection of *aggregate values* is computed; these values are typically related to the volume of the bin, such as the sum of the bytes fields for all records that match the key. The first aggregate value is called the primary aggregate value.

Once all the SiLK Flow records are read, **rwstats** sorts the bins by the primary aggregate value in either decreasing order (for a top-N list) or increasing order (for a bottom-N list). The ordering of bins that have the same primary aggregate value is arbitrary. The bins are printed as text, and the number of bins to print may be specified as a fixed value (e.g., print 10 bins), as a threshold (print bins whose byte count is greater than 400), or as a percentage of the total volume across all bins (print bins that contain at least 10% of all the packets).

The user must provide the **--fields** switch to select the flow attribute(s) (or field(s)) that comprise the key for each bin. The available fields are similar to those supported by **rwcut(1)**; see the description of the **--fields** switch in the OPTIONS section below for the details or run **rwstats** with the **--help-fields** switch. The list of fields may be extended by loading PySiLK files (see **silkpython(3)**) or plug-ins (**silk-plugin(3)**). The fields are printed in the order in which they occur in the **--fields** switch. The size of the key is limited to 256 octets. A larger key more quickly uses the available the memory and results in slower performance.

The aggregate value(s) to compute for each bin are also chosen by the user. As with the key fields, the user may extend the list of aggregate fields by using PySiLK or plug-ins. The preferred way to specify the aggregate fields is to use the **--values** switch; the aggregate fields are printed in the order they occur in the **--values** switch. If the user does not select any aggregate value(s), **rwstats** defaults to computing the number of flow records for each bin. As with the key fields, requesting more aggregate values slows performance.

In addition to computing the primary aggregate value for the flows in each bin, **rwstats** computes that aggregate value across all flow records. When printing the results, the output for each bin includes the ratio of the bin's aggregate value to the total aggregate value (displayed as a percentage). In addition, a cumulative percentage column is printed. When the primary aggregate value is a distinct count, the cumulative percentage may be greater than 100. The percentage columns contain a question mark when the primary aggregate value comes from a plug-in since **rwstats** does not know whether summing the aggregate values is reasonable. The display of the percentage columns may be suppressed by specifying **--no-percents**.

rwstats attempts to keep all key and aggregate value data in the computer's memory. If **rwstats** runs out of memory, the current key and aggregate value data is written to a temporary file. Once all input has been processed, the data from the temporary files is merged to produce the final output. By default, these temporary files are stored in the */tmp* directory. Because these files can be large, it is strongly recommended that */tmp not* be used as the temporary directory. To modify the temporary directory used by **rwstats**, provide the **--temp-directory** switch, set the SILK_TMPDIR environment variable, or set the TMPDIR environment variable.

rwstats may also run out of memory if the requested Top-N is too large.

The **--presorted-input** switch may allow **rwstats** to process data more efficiently by causing **rwstats** to assume the input has been previously sorted with the **rwsort(1)** command. With this switch, **rwstats** does not need large amounts of memory during the binning stage because it does not bin each flow; instead, it keeps a running summation for the bin. When the key changes, the bin's primary aggregate value is

compared with those of the current Top-N (or Bottom-N) to see if the new bin is a closer to the top (or bottom). For the output to be meaningful, **rwsort** and **rwstats** *must* be invoked with the same **--fields** value. When multiple input files are specified and **--presorted-input** is given, **rwstats** merge-sorts the flow records from the input files. **rwstats** usually runs faster if you do *not* include the **--presorted-input** switch when counting distinct IP addresses, even when reading sorted input. Finally, you may get unusual results with **--presorted-input** when the **--fields** switch contains multiple time-related key fields (`sTime`, `duration`, `eTime`), or when the time-related key is not the final key listed in **--fields**; see the NOTES section for details.

Protocol Statistics Description

Alternatively, **rwstats** can provide statistics for each of bytes, packets, and bytes-per-packet giving minima, maxima, quartile, and interval flow-counts across all flows or across a list of protocols specified by the user.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

Top-N Invocation

To compute a Top-N or Bottom-N list, the key field(s) must be specified. Normally the **--fields** switch is used to specify the key field(s), but for backward compatibility older switches may be specified (see the Legacy Switches section below).

--fields=*KEY*

KEY contains the list of flow attributes (a.k.a. fields or columns) that make up the key into which flows are binned. The columns are displayed in the order the fields are specified. Each field may be specified once only. *KEY* is a comma separated list of field-names, field-integers, and ranges of field-integers; a range is specified by separating the start and end of the range with a hyphen (`-`). Field-names are case insensitive. Example:

```
--fields=stime,10,1-5
```

There is no default value for the **--fields** switch.

The complete list of built-in fields that the SiLK tool suite supports follows, though note that not all fields are present in all SiLK file formats; when a field is not present, its value is 0.

sIP,1
: source IP address

dIP,2
: destination IP address

sPort,3
: source port for TCP and UDP, or equivalent

dPort,4
: destination port for TCP and UDP, or equivalent. See note at `iType`.

protocol,5
IP protocol

packets,pkts,6
packet count

bytes,7
byte count

flags,8
bit-wise OR of TCP flags over all packets

sTime,9
starting time of flow (seconds resolution). When the time-related fields `sTime`,`duration`,`eTime` are all in use, **rwstats** ignores the final time field when binning the records.

duration,10
duration of flow (seconds resolution). See note at `sTime,9`.

eTime,11
end time of flow (seconds resolution). See note at `sTime,9`.

sensor,12
name or ID of the sensor where the flow was collected

class,20
class assigned to the flow by **rwflowpack(8)**. Binning by `class` and/or `type` equates to binning by the integer value used internally to represent the class/type pair. When **--fields** contains `class` but not `type`, **rwstats**'s output may have multiple rows with the same value(s) for the key field(s).

type,21
type assigned to the flow by **rwflowpack(8)**. See note on previous entry.

iType
the ICMP type value for ICMP or ICMPv6 flows and empty (numerically zero) for non-ICMP flows. Internally, SiLK stores the ICMP type and code in the `dPort` field. To avoid getting very odd results, either do not use the `dPort` field when your key includes ICMP field(s) or be certain to include the `protocol` field as part of your key. This field was added in SiLK 3.8.1.

iCode
the ICMP code value for ICMP or ICMPv6 flows and empty for non-ICMP flows. See note at `iType`.

icmpTypeCode,25
equivalent to `iType`,`iCode` when used in **--fields**. This field may not be mixed with `iType` or `iCode`, and this field is deprecated as of SiLK 3.8.1. As of SiLK 3.8.1, `icmpTypeCode` may no longer be used as the argument to the `Distinct:` value field; the `dPort` field provides an equivalent result as long as the input is limited to ICMP flow records.

Many SiLK file formats do not store the following fields and their values are always 0; they are listed here for completeness:

in,13
router SNMP input interface or vlanId if packing tools were configured to capture it (see **sensor.conf(5)**)

out,14
router SNMP output interface or postVlanId

nhIP,15
router next hop IP

SiLK can store flows generated by enhanced collection software that provides more information than NetFlow v5. These flows may support some or all of these additional fields; for flows without this additional information, the field's value is always 0.

initialFlags,26
TCP flags on first packet in the flow

sessionFlags,27
bit-wise OR of TCP flags over all packets except the first in the flow

attributes,28
flow attributes set by the flow generator:

`S`
all the packets in this flow record are exactly the same size

`F`
flow generator saw additional packets in this flow following a packet with a FIN flag (excluding ACK packets)

`T`
flow generator prematurely created a record for a long-running connection due to a timeout. (When the flow generator **yaf(1)** is run with the **--silk** switch, it will prematurely create a flow and mark it with `T` if the byte count of the flow cannot be stored in a 32-bit value.)

`C`
flow generator created this flow as a continuation of long-running connection, where the previous flow for this connection met a timeout (or a byte threshold in the case of **yaf**).

Consider a long-running ssh session that exceeds the flow generator's *active* timeout. (This is the active timeout since the flow generator creates a flow for a connection that still has activity). The flow generator will create multiple flow records for this ssh session, each spanning some portion of the total session. The first flow record will be marked with a `T` indicating that it hit the timeout. The second through next-to-last records will be marked with `TC` indicating that this flow both timed out and is a continuation of a flow that timed out. The final flow will be marked with a `C`, indicating that it was created as a continuation of an active flow.

application,29
guess as to the content of the flow. Some software that generates flow records from packet data, such as **yaf**, will inspect the contents of the packets that make up a flow and use traffic signatures to label the content of the flow. SiLK calls this label the *application*; **yaf** refers to it as the *appLabel*. The application is the port number that is traditionally used for that type of traffic (see the */etc/services* file on most UNIX systems). For example, traffic that the flow generator recognizes as FTP will have a value of 21, even if that traffic is being routed through the standard HTTP/web port (80).

The following fields provide a way to label the IPs or ports on a record. These fields require external files to provide the mapping from the IP or port to the label:

sType,16
for the source IP address, the value 0 if the address is non-routable, 1 if it is internal, or 2 if it is routable and external. Uses the mapping file specified by the SILK_ADDRESS_TYPES environment variable, or the *address_types.pmap* mapping file, as described in **addrtype(3)**.

dType,17
as **sType** for the destination IP address

scc,18

for the source IP address, a two-letter country code abbreviation denoting the country where that IP address is located. Uses the mapping file specified by the SILK_COUNTRY_CODES environment variable, or the *country_codes.pmap* mapping file, as described in **ccfilter(3)**. The abbreviations are those defined by ISO 3166-1 (see for example https://www.iso.org/iso-3166-country-codes.html or https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2) or the following special codes: **--** N/A (e.g. private and experimental reserved addresses); **a1** anonymous proxy; **a2** satellite provider; **o1** other

dcc,19

as **scc** for the destination IP

src-*map-name*

label contained in the prefix map file associated with *map-name*. If the prefix map is for IP addresses, the label is that associated with the source IP address. If the prefix map is for protocol/port pairs, the label is that associated with the protocol and source port. See also the description of the **--pmap-file** switch below and the **pmapfilter(3)** manual page.

dst-*map-name*

as **src-*map-name*** for the destination IP address or the protocol and destination port.

sval

as **src-*map-name*** when no map-name is associated with the prefix map file

dval

as **dst-*map-name*** when no map-name is associated with the prefix map file

Finally, the list of built-in fields may be augmented by the run-time loading of PySiLK code or plug-ins written in C (also called shared object files or dynamic libraries), as described by the **--python-file** and **--plugin** switches.

--values=*VALUES*

When computing a Top-N or Bottom-N, all flows that have the same key field(s) are binned together. For each bin, one or more aggregate values are computed as specified by *VALUES*, a comma separated list of names. Names are case insensitive. The first entry in *VALUES* is the primary value, and it is used as the basis to compute the Top-N or Bottom-N. If the **--values** switch is not specified (and no legacy switch that sets values is specified), **rwstats** counts the number of flow records for each bin. The aggregate fields are printed in the order they occur in *VALUES*. The names of the built-in value fields follow. This list can be augmented through the use of PySiLK and plug-ins.

Records

Count the number of flow records that mapped to each bin.

Packets

Sum the number of packets across all records that mapped to each bin.

Bytes

Sum the number of bytes across all records that mapped to each bin.

sIP-Distinct

Count the number of distinct source IP addresses that were seen for each bin.

dIP-Distinct

Count the number of distinct destination IP addresses that were seen for each bin.

Distinct:*KEY_FIELD*

Count the number of distinct values for *KEY_FIELD*, where *KEY_FIELD* is any field that can be used as an argument to **--fields** except for `icmpTypeCode`. For example, `Distinct:sPort` counts the number of distinct source ports for each bin. When this aggregate value field is used, the specified *KEY_FIELD* may not be present in the argument to **--fields** (since the distinct count in that case is one).

--plugin=*PLUGIN*

Augment the list of key fields and/or aggregate value fields by using run-time loading of the plug-in (shared object) whose path is *PLUGIN*. The switch may be repeated to load multiple plug-ins. The creation of plug-ins is described in the **silk-plugin(3)** manual page. When *PLUGIN* does not contain a slash (`/`), **rwstats** attempts to find a file named *PLUGIN* in the directories listed in the FILES section. If **rwstats** finds the file, it uses that path. If *PLUGIN* contains a slash or if **rwstats** does not find the file, **rwstats** relies on your operating system's **dlopen(3)** call to find the file. When the SILK_PLUGIN_DEBUG environment variable is non-empty, **rwstats** prints status messages to the standard error as it attempts to find and open each of its plug-ins.

--pmap-file=*PATH*

--pmap-file=*MAPNAME*:*PATH*

Load the prefix map file located at *PATH* and create fields named src-*map-name* and dst-*map-name* where *map-name* is either the *MAPNAME* part of the argument or the map-name specified when the file was created (see **rwpmapbuild(1)**). If no map-name is available, **rwstats** names the fields `sval` and `dval`. Specify *PATH* as `-` or `stdin` to read from the standard input. The switch may be repeated to load multiple prefix map files, but each prefix map must use a unique map-name. The **--pmap-file** switch(es) must precede the **--fields** switch. See also **pmapfilter(3)**.

--pmap-column-width=*NUM*

When printing a label associated with a prefix map, this switch gives the maximum number of characters to use when displaying the textual value of the field.

--python-file=*PATH*

When the SiLK Python plug-in is used, **rwstats** reads the Python code from the file *PATH* to define additional fields that can be used as part of the key or as an aggregate value. This file should call **register_field()** for each field it wishes to define. For details and examples, see the **silkpython(3)** and **pysilk(3)** manual pages.

To determine the value of N for a Top-N (or Bottom-N) list, one of the following switches **must** be specified. The primary value may limit which switch may be specified.

--count=*COUNT*

Print the *COUNT* bins with the largest (or smallest) primary values. When *COUNT* is 0, all bins are printed. If *COUNT* is 0 and **rwstats** runs out of memory while attempting to sort all bins, **rwstats** prints the Top-N or Bottom-N bins using the amount of memory it was able to allocate. **rwstats** did not accept a value of 0 for *COUNT* prior to SiLK 3.12.0.

--threshold=*THESHOLD*

Print the bins where the primary value is greater-than (or less-than) the value *THESHOLD*. Using this switch when the primary value comes from a plug-in causes **rwstats** to exit with an error. If **rwstats** runs out of memory while locating all bins that meet the threshold, **rwstats** prints the Top-N or Bottom-N bins using the amount of memory it was able to allocate. *rwstats* did not accept a value of 0 for *THESHOLD* prior to SiLK 3.12.0.

--percentage=*PERCENT*

Print the bins where the primary value is greater-than (or less-than) *PERCENT* percent of the sum of the primary values across all bins. *PERCENT* may be a floating point value between 0.0 and 100.0 inclusive. To use this switch, the **--presorted-input** switch must not be present and the primary value must be `Bytes`, `Packets`, `Records`, or a distinct count. If **rwstats** runs out of memory while locating all bins that meet the percentage, **rwstats** prints the Top-N or Bottom-N using the amount of memory it was able to allocate. The value of *PERCENT* was required to be an integer prior to SiLK 3.12.0. Support for computing the percentages of distinct counts was added in SiLK 3.16.0.

To determine whether to compute the Top-N or the Bottom-N, specify one of the following switches. If neither switch is given, **--top** is assumed:

--top

Sort the bins in order of decreasing primary aggregate value. This is the default behavior.

--bottom

Sort the bins in order of increasing primary aggregate value.

Protocol Statistics Invocation

The following switches compute and print, for each of bytes, packets, and bytes per packet, the minimum value, the maximum value, quartiles, and a count of the number of flows that fall into each of one of ten intervals statistics. These switches may not be combined with the switches that produce Top-N or Bottom-N lists.

--overall-stats

Print intervals and quartiles across all flows that were read by **rwstats**.

--detail-proto-stats=*PROTO[,PROTO...]*

Print intervals and quartiles for each individual protocol listed as an argument. The argument should be a comma separated list of protocols or ranges of protocols: `1-6,17`. Specifying this option implies **--overall-stats**.

Miscellaneous Switches

The following switches are available when **rwstats** is running in either mode, though many only applicable to the Top-N mode.

--presorted-input

Cause **rwstats** to assume that it is reading sorted input; i.e., that **rwstats**'s input file(s) were generated by **rwsort(1)** using the *exact same* value for the **--fields** switch. When no distinct counts are being computed, **rwstats** can process its input without needing to write temporary files. When multiple input files are specified, **rwstats** merge-sorts the flow records from the input files. When using **--presorted-input** and computing a Top-N or Bottom-N, the **--percentage** limit cannot be used. See the NOTES section for issues that may occur when using **--presorted-input**.

--no-percents

For the Top-N invocation, do not print the percent-of-total and cumulative-percentage columns. These columns contain a question mark when the primary aggregate value comes from a plug-in, and this switch allows you to suppress them.

--ipv6-policy=*POLICY*

Determine how IPv4 and IPv6 flows are handled when SiLK has been compiled with IPv6 support. When the switch is not provided, the SILK_IPV6_POLICY environment variable is checked for a policy. If it is also unset or contains an invalid policy, the *POLICY* is **mix**. When SiLK has not been compiled with IPv6 support, IPv6 flows are always ignored, regardless of the value passed to this switch or in the SILK_IPV6_POLICY variable. The supported values for *POLICY* are:

ignore
Ignore any flow record marked as IPv6, regardless of the IP addresses it contains.

asv4
Convert IPv6 flow records that contain addresses in the ::ffff:0:0/96 netblock (that is, IPv4-mapped IPv6 addresses) to IPv4 and ignore all other IPv6 flow records.

mix
Process the input as a mixture of IPv4 and IPv6 flow records. When an IP address is used as part of the key or value, this policy is equivalent to **force**.

force
Convert IPv4 flow records to IPv6, mapping the IPv4 addresses into the ::ffff:0:0/96 netblock.

only
Process only flow records that are marked as IPv6 and ignore IPv4 flow records in the input.

--bin-time=*SECONDS*

--bin-time

Adjust the times in the key fields 'sTime' and 'eTime' to appear on *SECONDS*-second boundaries (the floor of the time is used). As of SiLK 3.17.0, *SECONDS* may be a fractional value of 0.001 or greater. By default, times appear on 1-second boundaries. When the switch is used but no argument is given, **rwstats** uses 60-second time bins.

--timestamp-format=*FORMAT*

Specify the format and/or timezone to use when printing timestamps. When this switch is not specified, the SILK_TIMESTAMP_FORMAT environment variable is checked for a default format and/or timezone. If it is empty or contains invalid values, timestamps are printed in the default format, and the timezone is UTC unless SiLK was compiled with local timezone support. *FORMAT* is a comma-separated list of a format and/or a timezone. The format is one of:

default
Print the timestamps as *YYYY/MM/DD*T*hh*:*mm*:*ss*.

iso
Print the timestamps as *YYYY-MM-DD hh*:*mm*:*ss*.

m/d/y
Print the timestamps as *MM/DD/YYYY hh*:*mm*:*ss*.

epoch
Print the timestamps as the number of seconds since 00:00:00 UTC on 1970-01-01.

When a timezone is specified, it is used regardless of the default timezone support compiled into SiLK. The timezone is one of:

utc
Use Coordinated Universal Time to print timestamps.

local
Use the TZ environment variable or the local timezone.

--epoch-time
Print timestamps as epoch time (number of seconds since midnight GMT on 1970-01-01). This switch is equivalent to **--timestamp-format=epoch**, it is deprecated as of SiLK 3.0.0, and it will be removed in the SiLK 4.0 release.

--ip-format=*FORMAT*
Specify how IP addresses are printed, where *FORMAT* is a comma-separated list of the arguments described below. When this switch is not specified, the SILK_IP_FORMAT environment variable is checked for a value and that format is used if it is valid. The default *FORMAT* is `canonical`. *Since SiLK 3.7.0.*

canonical
Print IP addresses in the canonical format. If the key only contains IPv4 addresses, use dot-separated decimal (`192.0.2.1`). Otherwise, use colon-separated hexadecimal (`2001:db8::1`) or a mixed IPv4-IPv6 representation for IPv4-mapped IPv6 addresses (the ::ffff:0:0/96 netblock, e.g., `::ffff:192.0.2.1`) and IPv4-compatible IPv6 addresses (the ::/96 netblock other than ::/127, e.g., `::192.0.2.1`).

no-mixed
Print IP addresses in the canonical format (`192.0.2.1` or `2001:db8::1`) but do not used the mixed IPv4-IPv6 representations. For example, use `::ffff:c000:201` instead of `::ffff:192.0.2.1`. *Since SiLK 3.17.0.*

decimal
Print IP addresses as integers in decimal format. For example, print `192.0.2.1` and `2001:db8::1` as `3221225985` and `42540766411282592856903984951653826561`, respectively.

hexadecimal
Print IP addresses as integers in hexadecimal format. For example, print `192.0.2.1` and `2001:db8::1` as `c00000201` and `20010db80000000000000000000000001`, respectively.

zero-padded
Make all IP address strings contain the same number of characters by padding numbers with leading zeros. For example, print `192.0.2.1` and `2001:db8::1` as `192.000.002.001` and `2001:0db8:0000:0000:0000:0000:0000:0001`, respectively. For IPv6 addresses, this setting implies `no-mixed`, so that `::ffff:192.0.2.1` is printed as `0000:0000:0000:0000:0000:ffff:c000:0201`. As of SiLK 3.17.0, may be combined with any of the above, including `decimal` and `hexadecimal`.

The following arguments modify certain IP addresses prior to printing. These arguments may be combined with the above formats.

map-v4
Change IPv4 addresses to IPv4-mapped IPv6 addresses (addresses in the ::ffff:0:0/96 netblock) prior to formatting. *Since SiLK 3.17.0.*

unmap-v6
When the key contains IPv6 addresses, change any IPv4-mapped IPv6 addresses (addresses in the ::ffff:0:0/96 netblock) to IPv4 addresses prior to formatting. *Since SiLK 3.17.0.*

The following argument is also available:

force-ipv6

Set *FORMAT* to `map-v4,no-mixed`.

--integer-ips

Print IP addresses as integers. This switch is equivalent to **--ip-format=decimal**, it is deprecated as of SiLK 3.7.0, and it will be removed in the SiLK 4.0 release.

--zero-pad-ips

Print IP addresses as fully-expanded, zero-padded values in their canonical form. This switch is equivalent to **--ip-format=zero-padded**, it is deprecated as of SiLK 3.7.0, and it will be removed in the SiLK 4.0 release.

--integer-sensors

Print the integer ID of the sensor rather than its name.

--integer-tcp-flags

Print the TCP flag fields (flags, initialFlags, sessionFlags) as an integer value. Typically, the characters `F,S,R,P,A,U,E,C` are used to represent the TCP flags.

--no-titles

Disable section and column titles. By default, titles are printed.

--no-columns

Disable fixed-width columnar output.

--column-separator=*C*

Use specified character between columns and after the final column. When this switch is not specified, the default of '|' is used.

--no-final-delimiter

Do not print the column separator after the final column. Normally a delimiter is printed.

--delimited

--delimited=*C*

Run as if **--no-columns --no-final-delimiter --column-sep**=*C* had been specified. That is, disable fixed-width columnar output; if character *C* is provided, it is used as the delimiter between columns instead of the default '|'.

--print-filenames

Print to the standard error the names of input files as they are opened.

--copy-input=*PATH*

Copy all binary SiLK Flow records read as input to the specified file or named pipe. *PATH* may be `stdout` or `-` to write flows to the standard output as long as the **--output-path** switch is specified to redirect **rwstats**' textual output to a different location.

--output-path=*PATH*

Write the textual output to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output (and bypass the paging program). If *PATH* names an existing file, **rwstats** exits with an error unless the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. If this switch is not given, the output is either sent to the pager or written to the standard output.

--pager=*PAGER_PROG*

When output is to a terminal, invoke the program *PAGER_PROG* to view the output one screen full at a time. This switch overrides the SILK_PAGER environment variable, which in turn overrides the PAGER variable. If the **--output-path** switch is given or if the value of the pager is determined to be the empty string, no paging is performed and all output is written to the terminal.

--temp-directory=*DIR_PATH*

Specify the name of the directory in which to store data files temporarily when the memory is not large enough to store all the bins and their aggregate values. This switch overrides the directory specified in the SILK_TMPDIR environment variable, which overrides the directory specified in the TMPDIR variable, which overrides the default, */tmp*.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwstats** searches for the site configuration file in the locations specified in the FILES section.

--legacy-timestamps

--legacy-timestamps=*NUM*

When *NUM* is not specified or is 1, this switch is equivalent to **--timestamp-format=m/d/y**. Otherwise, the switch has no effect. This switch is deprecated as of SiLK 3.0.0, and it will be removed in the SiLK 4.0 release.

--xargs

--xargs=*FILENAME*

Read the names of the input files from *FILENAME* or from the standard input if *FILENAME* is not provided. The input is expected to have one filename per line. **rwstats** opens each named file in turn and reads records from it as if the filenames had been listed on the command line.

--help

Print the available options and exit. Specifying switches that add new fields, values, or additional switches before **--help** allows the output to include descriptions of those fields or switches.

--help-fields

Print the description and alias(es) of each field and value and exit. Specifying switches that add new fields before **--help-fields** allows the output to include descriptions of those fields.

--legacy-help

Print help, including legacy switches. See the Legacy Switches section below for these switches.

--version

Print the version number and information about how SiLK was configured, then exit the application.

Legacy Switches

Use of the following switches has been discouraged since SiLK 2.0.0. As of SiLK 3.8.1, the switches are deprecated and they will be removed in SiLK 4.0. For each switch, use the replacement indicated.

--sip

Use: **--fields=sip**

--sip=*CIDR*

Use the most significant *CIDR* bits of the source address as the key. Using this switch with IPv6 data causes an error. The user should use **rwnetmask(1)** to mask the data prior to processing it with **rwstats**.

--dip

Use: **--fields=dip**

--dip=*CIDR*

Use the most significant *CIDR* bits of the destination address as the key. Using this switch with IPv6 data causes an error. The user should use **rwnetmask** to mask the data prior to processing it with **rwstats**.

--sport

Use: **--fields=sport**

--dport

Use: **--fields=dport**

--protocol

Use: **--fields=protocol**

--icmp

Use: **--fields=iType,iCode**

--flows

Use: `--values=records`

--packets

Use: `--values=packets`

--bytes

Use: `--values=bytes`

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

Top-N Examples

Print the top talkers (based on number of flow records, limit to the top four):

```
$ rwstats --fields=sip --count=4 data.rw
INPUT: 549092 Records for 12990 Bins and 549092 Total Records
OUTPUT: Top 4 Bins by Records
        sIP|     Records|  %Records|   cumul_%|
   10.1.1.1|       36604|  6.666278|  6.666278|
   10.1.1.2|       13897|  2.530906|  9.197184|
   10.1.1.3|       12739|  2.320012| 11.517196|
   10.1.1.4|       11807|  2.150277| 13.667473|
```

Print the seven hosts that received the most packets:

```
$ rwstats --fields=dip --values=packets --count=7 data.rw
INPUT: 549092 Records for 44654 Bins and 6620587 Total Packets
OUTPUT: Top 7 Bins by Packets
        dIP|     Packets|  %Packets|   cumul_%|
   10.1.1.1|      217574|  3.286325|  3.286325|
   10.1.1.2|      138177|  2.087081|  5.373407|
   10.1.1.3|      121892|  1.841106|  7.214512|
   10.1.1.4|       97073|  1.466230|  8.680742|
   10.1.1.5|       82284|  1.242851|  9.923593|
   10.1.1.6|       80051|  1.209123| 11.132715|
   10.1.1.7|       73602|  1.111714| 12.244430|
```

Print the IP pairs that shared 100,000,000 bytes or more:

```
$ rwstats --fields=sip,dip --values=byte --threshold=100000000 data.rw
INPUT: 549092 Records for 107136 Bins and 3410300252 Total Bytes
OUTPUT: Top 5 Bins by Bytes (threshold 100000000)
        sIP|            dIP|       Bytes|    %Bytes|   cumul_%|
   10.1.1.1|       10.1.1.2|   307478707|  9.016177|  9.016177|
   10.1.1.3|       10.1.1.4|   172164463|  5.048367| 14.064544|
   10.1.1.5|       10.1.1.6|   142059589|  4.165604| 18.230147|
   10.1.1.7|       10.1.1.8|   119388394|  3.500818| 21.730965|
   10.1.1.9|      10.1.1.10|   108268824|  3.174759| 24.905725|
```

Print the ports that were the source of at least 5% of all records:

```
$ rwstats --fields=sport --percentage=5 data.rw
INPUT: 549092 Records for 56799 Bins and 549092 Total Records
OUTPUT: Top 3 Bins by Records (5% == 27454)
  sPort|     Records|  %Records|   cumul_%|
     80|       86677| 15.785515| 15.785515|
     53|       64681| 11.779629| 27.565144|
      0|       47760|  8.697996| 36.263140|
```

Print the destination ports that saw the least number of records (limit to the bottom eight):

```
$ rwstats --fields=dport --bottom --count=8 data.rw
INPUT: 549092 Records for 44772 Bins and 549092 Total Records
OUTPUT: Bottom 8 Bins by Records
  dPort|     Records|  %Records|   cumul_%|
  19417|           1|  0.000182|  0.000182|
  12110|           1|  0.000182|  0.000364|
  34777|           1|  0.000182|  0.000546|
   8999|           1|  0.000182|  0.000728|
  36404|           1|  0.000182|  0.000911|
  16682|           1|  0.000182|  0.001093|
  27420|           1|  0.000182|  0.001275|
  14162|           1|  0.000182|  0.001457|
```

Print the source-destination port pairs that shared more than 500,000 packets (there were none):

```
$ rwstats --fields=sport,dport --values=packets \
       --top --threshold=500000 data.rw
INPUT: 366309 Records for 130307 Bins and 5597540 Total Packets
OUTPUT: No bins above threshold of 500000
```

Print the source-destination port pairs that shared more than 50,000 packets:

```
$ rwstats --fields=sport,dport --values=packets \
       --top --threshold=50000 data.rw
INPUT: 366309 Records for 130307 Bins and 5597540 Total Packets
OUTPUT: Top 3 Bins by Packets (threshold 50000)
  sPort|dPort|      Packets|  %Packets|   cumul_%|
   6699| 3607|       138177|  2.468531|  2.468531|
     80| 1179|        59774|  1.067862|  3.536393|
     80| 9659|        50319|  0.898949|  4.435342|
```

Print the protocols from least to most active (based on number of records):

```
$ rwstats --fields=protocol --bottom --count=10 data.rw
INPUT: 545262 Records for 3 Bins and 545262 Total Records
OUTPUT: Bottom 10 Bins by Records
   protocol|     Records|  %Records|   cumul_%|
          1|       46319|  8.494815|  8.494815|
         17|      132634| 24.324820| 32.819635|
          6|      366309| 67.180365|100.000000|
```

Print the packet and byte counts for the pair of /16s that shared the most packets (use **rwnetmask(1)** on the input to **rwstats**; limit result to top ten):

```
$ rwnetmask --4sip-prefix=16 --4dip-prefix=16 data.rw   \
  | rwstats --fields=sip,dip --values=packets,bytes     \
       --count=10 --no-percent
INPUT: 250928 Records for 230 Bins and 72279154 Total Packets
OUTPUT: Top 10 Bins by Packets
           sIP|            dIP|  Packets|        Bytes|
    10.255.0.0|    192.168.0.0|  2711524|   2207297227|
    10.253.0.0|    192.168.0.0|  2690120|   2288595669|
    10.254.0.0|    192.168.0.0|  2593074|   2141263178|
    10.252.0.0|    192.168.0.0|  2553388|   2117294828|
    10.250.0.0|    192.168.0.0|  2312661|   1982654956|
    10.251.0.0|    192.168.0.0|  2218194|   1785263601|
    10.249.0.0|    192.168.0.0|  2196041|   1934938137|
    10.248.0.0|    192.168.0.0|  2160037|   1804446929|
    10.247.0.0|    192.168.0.0|  2000379|   1579214987|
    10.246.0.0|    192.168.0.0|  1878143|   1578321728|
```

Print the number of distinct destination hosts seen for every destination port, limiting the result to the ports that saw at least 3% of the hosts. The percentage for each bin is relative to the number of distinct destination IP addresses seen in the input.

```
$ rwstats --fields=dport --values=distinct:dip --percent=3 data.rw
INPUT: 243127 Records for 4738 Bins and 122064 Total dIP-Distinct
OUTPUT: Top 5 bins by dIP-Distinct (3.0000% == 3661)
dPort|dIP-Distin|%dIP-Disti|   cumul_%|
   80|     26940| 22.070389|   22.0704|
   25|     15538| 12.729388|   34.7998|
  443|      7907|  6.477749|   41.2775|
   22|      7733|  6.335201|   47.6127|
 8080|      3942|  3.229453|   50.8422|
```

Print the number of distinct destination ports seen for each protocol. When the primary aggregate value is counting the number of distinct values, the cumulative percentage may be larger than 100%.

```
$ rwstats --fields=proto --values=distinct:dport --count=0 data.rw
INPUT: 243127 Records for 2 Bins and 5335 Total dPort-Distinct
OUTPUT: Top 2 Bins by dPort-Distinct
pro|dPort-Dist|%dPort-Dis|   cumul_%|
  6|      4672| 87.572634|   87.5726|
 17|      4669| 87.516401|  175.0890|
```

The following example uses PySiLK to create an aggregate value field that computes the average byte count for each bin. The code for this field is shown in the **silkpython(3)** manual page. Note that the percentage columns are empty.

```
$ rwstats --python-file=avg-bytes.py --fields=sport      \
       --values=avg-bytes,bytes,flow --count=6 data.rw
INPUT: 243127 Records for 4738 Bins
OUTPUT: Top 6 Bins by avg-bytes
sPort|    avg-bytes|         Bytes|   Records|%avg-bytes|   cumul_%|
   22|   1010658.57|   28292376134|     27994|         ?|         ?|
 8080|    739703.65|    2918870591|      3946|         ?|         ?|
   80|    732930.03|   19821359790|     27044|         ?|         ?|
  443|    731919.66|    5794607921|      7917|         ?|         ?|
25605|     86376.00|         86376|         1|         ?|         ?|
25349|     83556.00|        167112|         2|         ?|         ?|
```

The **--threshold** switch is not supported when the primary aggregate value is from PySiLK or a plug-in.

```
$ rwstats --python-file=avg-bytes.py --fields=sport            \
       --values=avg-bytes,bytes,flow --threshold=90000 data.rw
rwstats: Only the --count limit is supported when the primary \
       values field is from a plug-in
rwstats: Cannot add value field 'avg-bytes' from plugin
```

When using **rwstats** on input that contains both incoming and outgoing flow records, consider using the **int-ext-fields(3)** plug-in which defines four additional fields representing the external IP address, the external port, the internal IP address, and the internal port. The plug-in requires the user to specify which class/type pairs are incoming and which are outgoing. See its manual page for additional information. As an example, here we run **rwstats** on a file containing incoming and outgoing web traffic.

```
$ rwstats --fields=sip,sport,dip,dport --values=bytes \
       --count=6 data.rw
INPUT: 155140 Records for 155140 Bins and 59036553615 Total Bytes
OUTPUT: Top 6 Bins by Bytes
            sIP|sPort|            dIP|dPort|   Bytes|    %Bytes|   cumul_%|
  10.242.96.200|   80|192.168.234.203|29868| 2681287|  0.004542|  0.004542|
192.168.211.200|   80|  10.253.27.160|25453| 2675740|  0.004532|  0.009074|
192.168.233.168|   80|  10.247.60.163|29777| 2672196|  0.004526|  0.013600|
192.168.229.229|  443|  10.250.19.210|27512| 2666647|  0.004517|  0.018117|
 192.168.255.24| 8080|  10.240.75.236|29826| 2659828|  0.004505|  0.022623|
192.168.241.247|   80|  10.216.173.77|26654| 2658141|  0.004503|  0.027125|
```

Here the **int-ext-fields** plug-in is used:

```
$ export INCOMING_FLOWTYPES=all/in,all/inweb
$ export OUTGOING_FLOWTYPES=all/out,all/outweb
$ rwstats --plugin=int-ext-fields.so                          \
       --fields=ext-ip,ext-port,int-ip,int-port --value=bytes \
       --count=6 data.rw
INPUT: 155140 Records for 77570 Bins and 59036553615 Total Bytes
OUTPUT: Top 6 Bins by Bytes
         ext-ip|ext-p|         int-ip|int-p|   Bytes|    %Bytes|   cumul_%|
  10.253.27.160|25453|192.168.211.200|   80| 2736332|  0.004635|  0.004635|
  10.242.96.200|   80|192.168.234.203|29868| 2722619|  0.004612|  0.009247|
  10.247.60.163|29777|192.168.233.168|   80| 2716749|  0.004602|  0.013849|
  10.250.19.210|27512|192.168.229.229|  443| 2714974|  0.004599|  0.018447|
  10.254.241.55|24206| 192.168.207.45|   80| 2713597|  0.004596|  0.023044|
 10.226.206.118|29557|192.168.247.227| 8080| 2707265|  0.004586|  0.027630|
```

Protocol Statistics Example

Print the interval breakdowns for flow records, packets, and bytes across all protocols, and for protocols 6 (TCP) and 17 (UDP):

```
$ rwstats --detail-proto-stats=6,17 data.rw
FLOW STATISTICS--ALL PROTOCOLS:  549092 records
*BYTES min 28; max 88906238
  quartiles LQ 122.06478 Med 420.30930 UQ 876.21920 UQ-LQ 754.15442
   interval_max|count<=max|%_of_input|   cumul_%|
             40|     35107|  6.393646|  6.393646|
             60|     35008|  6.375616| 12.769263|
            100|     49500|  9.014883| 21.784145|
            150|     40014|  7.287303| 29.071449|
            256|     65444| 11.918586| 40.990034|
           1000|    224016| 40.797535| 81.787569|
          10000|     75708| 13.787853| 95.575423|
         100000|     21981|  4.003154| 99.578577|
        1000000|      1901|  0.346208| 99.924785|
     4294967295|       413|  0.075215|100.000000|
*PACKETS min 1; max 70023
  quartiles LQ 1.76962 Med 3.68119 UQ 7.61567 UQ-LQ 5.84605
```

```
   interval_max|count<=max|%_of_input|   cumul_%|
              3|    232716| 42.381969| 42.381969|
              4|     61407| 11.183372| 53.565341|
             10|    195310| 35.569631| 89.134972|
             20|     33310|  6.066379| 95.201351|
             50|     17686|  3.220954| 98.422304|
            100|      4854|  0.884005| 99.306309|
            500|      2760|  0.502648| 99.808957|
           1000|       373|  0.067930| 99.876888|
          10000|       637|  0.116010| 99.992897|
     4294967295|        39|  0.007103|100.000000|
*BYTES/PACKET min 28; max 1500
  quartiles LQ 57.98319 Med 90.71150 UQ 164.77250 UQ-LQ 106.78932
   interval_max|count<=max|%_of_input|   cumul_%|
             40|     42568|  7.752435|  7.752435|
             44|     15173|  2.763289| 10.515724|
             60|     91003| 16.573361| 27.089085|
            100|    163850| 29.840173| 56.929258|
            200|    153190| 27.898786| 84.828043|
            400|     39761|  7.241227| 92.069271|
            600|     12810|  2.332942| 94.402213|
            800|      7954|  1.448573| 95.850786|
           1500|     22783|  4.149214|100.000000|
     4294967295|         0|  0.000000|100.000000|

FLOW STATISTICS--PROTOCOL 6:  366309/549092 records
*BYTES min 40; max 88906238
  quartiles LQ 310.47331 Med 656.53661 UQ 1089.75344 UQ-LQ 779.28013
   interval_max|count<=max|%_of_proto|   cumul_%|
             40|     29774|  8.128110|  8.128110|
             60|     11453|  3.126595| 11.254706|
            100|      6915|  1.887751| 13.142456|
            150|     16369|  4.468632| 17.611088|
            256|     12651|  3.453642| 21.064730|
           1000|    196881| 53.747246| 74.811976|
          10000|     68989| 18.833553| 93.645529|
         100000|     21099|  5.759891| 99.405420|
        1000000|      1784|  0.487021| 99.892441|
     4294967295|       394|  0.107559|100.000000|
*PACKETS min 1; max 70023
  quartiles LQ 3.39682 Med 5.85903 UQ 8.80427 UQ-LQ 5.40745
   interval_max|count<=max|%_of_proto|   cumul_%|
              3|     69358| 18.934288| 18.934288|
              4|     55993| 15.285729| 34.220016|
             10|    186559| 50.929407| 85.149423|
             20|     30947|  8.448332| 93.597755|
             50|     16186|  4.418674| 98.016429|
            100|      4204|  1.147665| 99.164094|
            500|      2178|  0.594580| 99.758674|
           1000|       315|  0.085993| 99.844667|
          10000|       537|  0.146598| 99.991264|
     4294967295|        32|  0.008736|100.000000|
```

```
*BYTES/PACKET min 40; max 1500
  quartiles LQ 60.19817 Med 96.78616 UQ 175.08044 UQ-LQ 114.88228
   interval_max|count<=max|%_of_proto|   cumul_%|
             40|     36559|  9.980372|  9.980372|
             44|     14929|  4.075521| 14.055893|
             60|     39593| 10.808634| 24.864527|
            100|    100117| 27.331297| 52.195824|
            200|    111258| 30.372718| 82.568542|
            400|     26020|  7.103293| 89.671834|
            600|      8600|  2.347745| 92.019579|
            800|      7726|  2.109148| 94.128727|
           1500|     21507|  5.871273|100.000000|
     4294967295|         0|  0.000000|100.000000|

FLOW STATISTICS--PROTOCOL 17:  132634/549092 records
*BYTES min 32; max 2115559
  quartiles LQ 66.53665 Med 150.61551 UQ 242.44095 UQ-LQ 175.90430
   interval_max|count<=max|%_of_proto|   cumul_%|
             20|         0|  0.000000|  0.000000|
             40|      5195|  3.916794|  3.916794|
             80|     42150| 31.779182| 35.695975|
            130|     11528|  8.691587| 44.387563|
            256|     45497| 34.302667| 78.690230|
           1000|     23401| 17.643289| 96.333519|
          10000|      4447|  3.352836| 99.686355|
         100000|       389|  0.293288| 99.979643|
        1000000|        23|  0.017341| 99.996984|
     4294967295|         4|  0.003016|100.000000|
*PACKETS min 1; max 8839
  quartiles LQ 0.84383 Med 1.68768 UQ 2.53149 UQ-LQ 1.68766
   interval_max|count<=max|%_of_proto|   cumul_%|
              3|    117884| 88.879171| 88.879171|
              4|      4452|  3.356605| 92.235777|
             10|      6678|  5.034908| 97.270685|
             20|      1766|  1.331484| 98.602168|
             50|      1055|  0.795422| 99.397590|
            100|       368|  0.277455| 99.675046|
            500|       353|  0.266146| 99.941192|
           1000|        33|  0.024880| 99.966072|
          10000|        45|  0.033928|100.000000|
     4294967295|         0|  0.000000|100.000000|
*BYTES/PACKET min 32; max 1415
  quartiles LQ 63.23827 Med 91.27180 UQ 158.10219 UQ-LQ 94.86392
   interval_max|count<=max|%_of_proto|   cumul_%|
             20|         0|  0.000000|  0.000000|
             24|         0|  0.000000|  0.000000|
             40|      5671|  4.275676|  4.275676|
            100|     70970| 53.508150| 57.783826|
            200|     39298| 29.628904| 87.412730|
            400|     12175|  9.179396| 96.592126|
            600|      4130|  3.113832| 99.705958|
            800|       160|  0.120633| 99.826590|
```

```
      1500|       230|  0.173410|100.000000|
4294967295|         0|  0.000000|100.000000|
```

The **silkpython(3)** manual page provides examples that use PySiLK to create arbitrary fields to use as part of the key for **rwstats**.

ENVIRONMENT

SILK_IPV6_POLICY

This environment variable is used as the value for **--ipv6-policy** when that switch is not provided.

SILK_IP_FORMAT

This environment variable is used as the value for **--ip-format** when that switch is not provided. *Since SiLK 3.11.0.*

SILK_TIMESTAMP_FORMAT

This environment variable is used as the value for **--timestamp-format** when that switch is not provided. *Since SiLK 3.11.0.*

SILK_PAGER

When set to a non-empty string, **rwstats** automatically invokes this program to display its output a screen at a time. If set to an empty string, **rwstats** does not automatically page its output.

PAGER

When set and SILK_PAGER is not set, **rwstats** automatically invokes this program to display its output a screen at a time.

SILK_TMPDIR

When set and **--temp-directory** is not specified, **rwstats** writes the temporary files it creates to this directory. SILK_TMPDIR overrides the value of TMPDIR.

TMPDIR

When set and SILK_TMPDIR is not set, **rwstats** writes the temporary files it creates to this directory.

PYTHONPATH

This environment variable is used by Python to locate modules. When **--python-file** is specified, **rwstats** must load the Python files that comprise the PySiLK package, such as *silk/__init__.py*. If this *silk/* directory is located outside Python's normal search path (for example, in the SiLK installation tree), it may be necessary to set or modify the PYTHONPATH environment variable to include the parent directory of *silk/* so that Python can find the PySiLK module.

SILK_PYTHON_TRACEBACK

When set, Python plug-ins output traceback information on Python errors to the standard error.

SILK_COUNTRY_CODES

This environment variable allows the user to specify the country code mapping file that **rwstats** uses when computing the scc and dcc fields. The value may be a complete path or a file relative to the SILK_PATH. See the FILES section for standard locations of this file.

SILK_ADDRESS_TYPES

This environment variable allows the user to specify the address type mapping file that **rwstats** uses when computing the sType and dType fields. The value may be a complete path or a file relative to the SILK_PATH. See the FILES section for standard locations of this file.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwstats** may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files and plug-ins, **rwstats** may use this environment variable. See the FILES section for details.

TZ

When the argument to the **--timestamp-format** switch includes `local` or when a SiLK installation is built to use the local timezone, the value of the TZ environment variable determines the timezone in which **rwstats** displays timestamps. (If both of those are false, the TZ environment variable is ignored.) If the TZ environment variable is not set, the machine's default timezone is used. Setting TZ to the empty string or 0 causes timestamps to be displayed in UTC. For system information on the TZ variable, see **tzset(3)** or **environ(7)**. (To determine if SiLK was built with support for the local timezone, check the `Timezone support` value in the output of **rwstats --version**.)

SILK_PLUGIN_DEBUG

When set to 1, **rwstats** prints status messages to the standard error as it attempts to find and open each of its plug-ins. In addition, when an attempt to register a field fails, **rwstats** prints a message specifying the additional function(s) that must be defined to register the field in **rwstats**. Be aware that the output can be rather verbose.

SILK_TEMPFILE_DEBUG

When set to 1, **rwstats** prints debugging messages to the standard error as it creates, re-opens, and removes temporary files.

SILK_UNIQUE_DEBUG

When set to 1, the binning engine used by **rwstats** prints debugging messages to the standard error.

FILES

${SILK_ADDRESS_TYPES}

${SILK_PATH}/share/silk/address_types.pmap

${SILK_PATH}/share/address_types.pmap

/usr/local/share/silk/address_types.pmap

/usr/local/share/address_types.pmap

Possible locations for the address types mapping file required by the sType and dType fields.

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

${SILK_COUNTRY_CODES}

${SILK_PATH}/share/silk/country_codes.pmap

${SILK_PATH}/share/country_codes.pmap

/usr/local/share/silk/country_codes.pmap

/usr/local/share/country_codes.pmap

Possible locations for the country code mapping file required by the scc and dcc fields.

${SILK_PATH}/lib64/silk/

${SILK_PATH}/lib64/

${SILK_PATH}/lib/silk/

${SILK_PATH}/lib/

/usr/local/lib64/silk/

/usr/local/lib64/

/usr/local/lib/silk/

/usr/local/lib/

Directories that **rwstats** checks when attempting to load a plug-in.

${SILK_TMPDIR}/

${TMPDIR}/

/tmp/

Directory in which to create temporary files.

NOTES

rwstats functionally replaces the combination the following, where N is one more than the number of fields passed to **rwuniq(1)**:

```
rwuniq --fields=... | sort -r -t '|' -k N | head -10
```

When the **--bin-time** switch is given and the three time fields (starting-time (`sTime`), ending-time (`eTime`), and duration (`duration`)) are present in the key, the duration field's value is modified to be the difference between the ending and starting times.

When the three time-related key fields (`sTime`,`duration`,`eTime`) are all in use, **rwstats** ignores the final time field when binning the records, but the field does appear in the output. Due to truncation of the milliseconds values, **rwstats** generates different numbers of bins depending on the order in which those three values appear in the **--fields** switch.

When computing distinct counts over a field, the field may not be part of the key; that is, you may not have `--fields=sip --values=sip-distinct`. The distinct count in that case is always 1.

Using the **--presorted-input** switch sometimes introduces more issues than it solves, and **--presorted-input** is less necessary now that **rwstats** can use temporary files while processing input.

When computing distinct IP counts, **rwstats** typically runs faster if you do *not* use the **--presorted-input** switch, even if the data was previously sorted.

When using the **--presorted-input** switch, it is highly recommended that you use no more than one time-related key field (`sTime`, `duration`, `eTime`) in the **--fields** switch and that the time-related key appear last in **--fields**. The issue is caused by **rwsort** considering the millisecond values on the times when sorting, while **rwstats** truncates the millisecond value.

rwstats's strength is its ability to build arbitrary keys and aggregate fields. For maps of a single key to a single value, see also **rwbag(1)**. To create a binary file that contains multiple keys and values, use **rwaggbag(1)**.

SEE ALSO

rwcut(1), **rwnetmask(1)**, **rwsort(1)**, **rwuniq(1)**, **rwbag(1)**, **rwaggbag(1)**, **rwpmapbuild(1)**, **addrtype(3)**, **ccfilter(3)**, **int-ext-fields(3)**, **pmapfilter(3)**, **pysilk(3)**, **silkpython(3)**, **silk-plugin(3)**, **sensor.conf(5)**, **rwflowpack(8)**, **silk(7)**, **yaf(1)**, **dlopen(3)**, **tzset(3)**, **environ(7)**

rwswapbytes

Change the byte order of a SiLK Flow file

SYNOPSIS

```
rwswapbytes
      { --big-endian | --little-endian
        | --native-endian | --swap-endian }
      [--note-add=TEXT] [--note-file-add=FILE]
      [INPUT_FILE [OUTPUT_FILE]]

rwswapbytes --help

rwswapbytes --version
```

DESCRIPTION

Read the SiLK Flow records from *INPUT_FILE*, change the byte order of each record as specified by the **--big-endian**, **--little-endian**, **--native-endian**, or **--swap-endian** switch, and write the records to *OUTPUT_FILE*.

rwswapbytes reads the input from the standard input either when no non-switch arguments are given or when *INPUT_FILE* is the string `stdin` or `-`. **rwswapbytes** writes the output to the standard output either when the number of non-switch arguments is less than two or when *OUTPUT_FILE* is the string `stdout` or `-`.

rwswapbytes exits with an error code if an attempt is made to read or write binary data from or to a terminal.

rwswapbytes is able to read and write files that have been compressed with **gzip(1)** when the file's name ends with `.gz`.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

One of these switches must be provided:

--big-endian

Write the output file in big-endian (network byte-order) format.

--little-endian

Write the output file in little-endian (Intel) format.

--native-endian

Write the output file in this machine's native format.

--swap-endian

Unconditionally swap the byte-order of the input file.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

These switches are optional:

--note-add=*TEXT*

Add the specified *TEXT* to the header of the output file as an annotation. This switch may be repeated to add multiple annotations to a file. To view the annotations, use the **rwfileinfo(1)** tool.

--note-file-add=*FILENAME*

Open *FILENAME* and add the contents of that file to the header of the output file as an annotation. This switch may be repeated to add multiple annotations. Currently the application makes no effort to ensure that *FILENAME* contains text; be careful that you do not attempt to add a SiLK data file as an annotation.

ENVIRONMENT

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_CONFIG_FILE

This environment variable contains the location of the site configuration file, **silk.conf(5)**. For additional locations where site configuration file may reside, see the FILES section.

SILK_DATA_ROOTDIR

This variable specifies the root of the directory tree where the data store of SiLK Flow files is maintained, overriding the location that is compiled into the tools (/data). **rwswapbytes** may search for the site configuration file, *silk.conf*, in this directory. See the FILES section for details.

SILK_PATH

This environment variable gives the root of the directory tree where the tools are installed. As part of its search for the site configuration file, **rwswapbytes** may use this variable. See the FILES section for details.

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file.

SEE ALSO

rwfileinfo(1), **silk.conf(5)**, **silk(7)**, **gzip(1)**

NOTES

Prior to SiLK 3.16.0, **rwswapbytes** required explicit arguments for the input file and the output file.

rwtotal

Count how much traffic matched specific keys

SYNOPSIS

```
rwtotal {--sip-first-8 | --sip-first-16 | --sip-first-24 |
      --sip-last-8 | --sip-last-16 | --dip-first-8 |
      --dip-first-16 | --dip-first-24 | --dip-last-8 |
      --dip-last-16 | --sport | --dport | --proto | --packets |
      --bytes | --duration | --icmp-code}
      [--summation] [--min-bytes=COUNT] [--max-bytes=COUNT]
      [--min-packets=COUNT] [--max-packets=COUNT]
      [--min-records=COUNT] [--max-records=COUNT] [--skip-zeroes]
      [--no-titles] [--no-columns] [--column-separator=CHAR]
      [--no-final-delimiter] [{--delimited | --delimited=CHAR}]
      [--print-filenames] [--copy-input=PATH] [--output-path=PATH]
      [--pager=PAGER_PROG] [--site-config-file=FILENAME]
      {[--xargs] | [--xargs=FILENAME] | [FILE [FILE ...]]}

rwtotal --help

rwtotal --version
```

DESCRIPTION

rwtotal reads SiLK Flow records, bins those records by the user-specified specified key, computes the volume per bin (record count and sums of packets and bytes), and prints the bins and their volumes.

rwtotal reads SiLK Flow records from the files named on the command line or from the standard input when no file names are specified and **--xargs** is not present. To read the standard input in addition to the named files, use `-` or `stdin` as a file name. If an input file name ends in `.gz`, the file is uncompressed as it is read. When the **--xargs** switch is provided, **rwtotal** reads the names of the files to process from the named text file or from the standard input if no file name argument is provided to the switch. The input to **--xargs** must contain one file name per line.

By default, **rwtotal** prints a bin for every possible key, even when the volume for that bin is zero. Use the **--skip-zeroes** switch to suppress the printing of these empty bins.

Use the **--summation** switch to include a row giving the volume for all flow records.

The maximum key value that **rwtotal** supports is 16,777,215. When the key field is **--bytes** or **--packets**, **rwtotal** will create a bin for all unique values up to 16,777,214. The final bin (16,777,215) will consist of all values greater than 16,777,214.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

One and only one of the following counting keys is required:

--sip-first-8
Key on the first 8 bits of the source IP address

--sip-first-16
Key on the first 16 bits of the source IP address

--sip-first-24
Key on the first 24 bits of the source IP address

--sip-last-8
Key on the last 8 bits of the source IP address

--sip-last-16
Key on the last 16 bits of the source IP address

--dip-first-8
Key on the first 8 bits of the destination IP address

--dip-first-16
Key on the first 16 bits of the destination IP address

--dip-first-24
Key on the first 24 bits of the destination IP address

--dip-last-8
Key on the last 8 bits of the destination IP address

--dip-last-16
Key on the last 16 bits of the destination IP address

--sport
Key on the source port.

--dport
Key on the destination port.

--proto
Key on the protocol.

--packets
Key on the number of packets in the record

--bytes
Key on the number of bytes in the record

--duration
Key on the duration of the record.

--icmp-code
Key on the ICMP type and code. This switch will assume that all incoming records are ICMP.

The following options affect the output:

--summation

Print as the final row a total of the values in each column.

--min-bytes=*COUNT*

Disable printing of bins with fewer than *COUNT* bytes. By default, all bins are printed.

--max-bytes=*COUNT*

Disable printing of bins with more than *COUNT* bytes. By default, all bins are printed.

--min-packets=*COUNT*

Disable printing of bins with fewer than *COUNT* packets. By default, all bins are printed.

--max-packets=*COUNT*

Disable printing of bins with more than *COUNT* packets. By default, all bins are printed.

--min-records=*COUNT*

Disable printing of bins with fewer than *COUNT* flow records. By default, all bins are printed.

--max-records=*COUNT*

Disable printing of bins with more than *COUNT* flow records. By default, all bins are printed.

--skip-zeroes

Disable printing of bins with no traffic. By default, all bins are printed.

--no-titles

Turn off column titles. By default, titles are printed.

--no-columns

Disable fixed-width columnar output.

--column-separator=*C*

Use specified character between columns and after the final column. When this switch is not specified, the default of '|' is used.

--no-final-delimiter

Do not print the column separator after the final column. Normally a delimiter is printed.

--delimited

--delimited=*C*

Run as if **--no-columns --no-final-delimiter --column-sep**=*C* had been specified. That is, disable fixed-width columnar output; if character *C* is provided, it is used as the delimiter between columns instead of the default '|'.

--print-filenames

Print to the standard error the names of input files as they are opened.

--copy-input=*PATH*

Copy all binary SiLK Flow records read as input to the specified file or named pipe. *PATH* may be `stdout` or `-` to write flows to the standard output as long as the **--output-path** switch is specified to redirect **rwtotal**'s textual output to a different location.

--output-path=*PATH*

Write the textual output to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output (and bypass the paging program). If *PATH* names an existing file, **rwtotal** exits with an error unless the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. If this switch is not given, the output is either sent to the pager or written to the standard output.

--pager=*PAGER_PROG*

When output is to a terminal, invoke the program *PAGER_PROG* to view the output one screen full at a time. This switch overrides the SILK_PAGER environment variable, which in turn overrides the PAGER variable. If the **--output-path** switch is given or if the value of the pager is determined to be the empty string, no paging is performed and all output is written to the terminal.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwtotal** searches for the site configuration file in the locations specified in the FILES section.

--xargs

--xargs=*FILENAME*

Read the names of the input files from *FILENAME* or from the standard input if *FILENAME* is not provided. The input is expected to have one filename per line. **rwtotal** opens each named file in turn and reads records from it as if the filenames had been listed on the command line.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

Group by the protocol

Group all incoming data for the first hour of March 1, 2003 by protocol.

```
$ rwfilter --start-date=2003/03/01:00 --end-date=2003/03/01:00 \
      --all-destination=stdout                                 \
  | rwtotal --proto --skip-zero
   protocol|        Records|               Bytes|          Packets|
          1|          15622|            10695328|           147084|
          6|         330726|        120536195111|        144254362|
         17|         155528|            24500079|           155528|
```

To get the same result with **rwuniq(1)**, use:

```
$ rwfilter ... --pass=stdout                                    \
  | rwuniq --fields=proto --values=records,bytes,packets        \
       --sort-output
pro|   Records|               Bytes|        Packets|
  1|     15622|            10695328|         147084|
  6|    330726|        120536195111|      144254362|
 17|    155528|            24500079|         155528|
```

Group by the source Class A addresses

```
$ rwfilter --start-date=2003/03/01:00 --end-date=2003/03/01:00 \
       --all-destination=stdout                                 \
  | rwtotal --sip-first-8 --skip-zero
 sIP_First8|        Records|               Bytes|          Packets|
         10|         173164|         59950837766|         72201390|
        172|          77764|            17553593|            77764|
        192|         250948|         60602999159|         72277820|
```

Use **rwnetmask(1)** and **rwuniq(1)** to get a similar result:

```
$ rwfilter ... --pass=stdout                                    \
  | rwnetmask --4sip-prefix=8                                   \
  | rwuniq --fields=sip --values=records,bytes,packets          \
       --sort-output --ipv6-policy=ignore
            sIP|   Records|               Bytes|        Packets|
       10.0.0.0|    173164|         59950837766|       72201390|
      172.0.0.0|     77764|            17553593|          77764|
      192.0.0.0|    250948|         60602999159|       72277820|
```

Group by the final IPv4 octet

```
$ rwfilter --start-date=2003/03/01:00 --end-date=2003/03/01:00     \
       --proto=6 --pass=stdout --daddress=192.168.x.x              \
  | rwtotal --dip-last-16 --skip-zero | head -5
 dIP_Last16|        Records|               Bytes|          Packets|
      0. 38|              6|             4862678|             4016|
      1. 14|              1|               32844|              452|
     18.146|              1|                4226|               12|
     21.  4|              6|             5462032|             4521|
```

One way to accomplish this with **rwuniq** is to create a new field using PySiLK (see **pysilk(3)**) and the PySiLK plug-in capability (see **silkpython(3)**. The invocation is:

```
$ rwfilter ... --pass=stdout                                      \
  | rwuniq --python=/tmp/dip16.py --fields=dip-last-16            \
       --values=flows,bytes,packets --sort-output | head -5
    dip-last-16|   Records|               Bytes|        Packets|
       0.0.0.38|         6|             4862678|           4016|
       0.0.1.14|         1|               32844|            452|
```

```
     0.0.18.146|         1|              4226|           12|
       0.0.21.4|         6|           5462032|         4521|
```

where the definition of the `dip-last-16` field is given in the file *tmp/dip16.py*:

```
import silk
mask = silk.IPAddr("0.0.255.255")
def mask_dip(r):
    return r.dip.mask(mask)

register_ipv4_field("dip-last-16", mask_dip)
```

ENVIRONMENT

SILK_PAGER

When set to a non-empty string, **rwtotal** automatically invokes this program to display its output a screen at a time. If set to an empty string, **rwtotal** does not automatically page its output.

PAGER

When set and SILK_PAGER is not set, **rwtotal** automatically invokes this program to display its output a screen at a time.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwtotal** may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **rwtotal** may use this environment variable. See the FILES section for details.

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

SEE ALSO

rwaddrcount(1), **rwnetmask(1)**, **rwstats(1)**, **rwuniq(1)**, **pysilk(3)**, **silkpython(3)**, **silk(7)**

BUGS

rwtotal replicates some functionality in **rwuniq(1)** (most notably when **rwuniq** checks by port or protocol), but the implementations differ: **rwtotal** uses an array instead of a hash-table, so access is faster, the output is always sorted, and the output includes keys with a value of zero. The use of an array prevents **rwtotal** from using the complete IP address the way **rwuniq** does, but it also ensures that **rwtotal** will not run out of memory.

When used in an IPv6 environment, **rwtotal** will process every record as long as the IP address is not part of the key. When aggregating by the IP address, **rwtotal** converts IPv6 flow records that contain addresses in the ::ffff:0:0/96 prefix to IPv4 and processes them. IPv6 records having addresses outside of that prefix are silently ignored. **rwtotal** will not be modified to support IPv6 addresses; instead, users should use **rwuniq(1)** (maybe combined with **rwnetmask(1)**).

rwtotal is also similar to **rwaddrcount(1)** and **rwstats(1)**.

rwtuc

Text Utility Converter - **rwcut** output to SiLK flows

SYNOPSIS

```
rwtuc [--fields=FIELDS] [--column-separator=CHAR]
      [--output-path=PATH] [--bad-input-lines=FILEPATH]
      [--verbose] [--stop-on-error] [--no-titles] [--note-add=TEXT]
      [--note-file-add=FILE] [--compression-method=COMP_METHOD]
      [--site-config-file=FILENAME] [--saddress=IPADDR]
      [--daddress=IPADDR] [--sport=NUM] [--dport=NUM]
      [--protocol=NUM] [--packets=NUM] [--bytes=NUM]
      [--flags-all=TCPFLAGS] [--stime=TIME] [--duration=NUM]
      [--etime=TIME] [--sensor=SID] [--input-index=NUM]
      [--output-index=NUM] [--next-hop-ip=IPADDR]
      [--flags-initial=TCPFLAGS] [--flags-session=TCPFLAGS]
      [--attributes=ATTR] [--application=NUM] [--class=NAME]
      [--type=NAME] [--stime+msec=TIME] [--etime+msec=TIME]
      [--duration+msec=NUM] [--icmp-type=NUM] [--icmp-code=NUM]
      {[--xargs] | [--xargs=FILENAME] | [FILE [FILE...]]}

rwtuc --help

rwtuc --version
```

DESCRIPTION

rwtuc reads text files that have a format similar to that produced by **rwcut(1)** and attempts to create a SiLK Flow record for each line of input.

The fields which make up a single record should be separated by the pipe character ('|'); use the **--column-separator** switch to change this delimiter. Note that the space character does not work as delimiter since several fields (e.g., time, TCP-flags) may contain embedded spaces.

The fields to be read from each line may be specified with the **--fields** switch; if the switch is not provided, **rwtuc** treats the first line as a title and attempts to determine the fields from the title strings.

When **--fields** is specified, **rwtuc** still checks whether the first line contains title strings, and **rwtuc** skips the line if it determines it does. Specify the **--no-titles** switch to force **rwtuc** to treat the first line as field values to be parsed.

Command line switches exist which force a field to have a fixed value. These switches cause **rwtuc** to override the value read from the input file (if any) for those fields. See the Fixed Values section below for details.

rwtuc reads the textual input from the files named on the command line or from the standard input when no file names are specified, when **--xargs** is not present, and when the standard input is not a terminal. To read the standard input in addition to the named files or to force **rwfileinfo** to read input from a terminal, use `-` or `stdin` as a file name. When the **--xargs** switch is provided, **rwtuc** reads the names of the files to process from the named text file or from the standard input if no file name argument is provided to the switch. The input to **--xargs** must contain one file name per line.

When the **--output-path** switch is not provided, output is sent to the standard output when it is not connected to a terminal.

By default, lines that cannot be parsed are silently ignored (unless **rwtuc** is attempting to determine the fields from the title line). When the **--verbose** switch is specified, problems parsing an input line are reported to the standard error, and **rwtuc** continues to process the input. The **--stop-on-error** switch is similar to the **--verbose** switch, except processing stops after the first error. Input lines that cause parse errors may be copied to another output stream with the **--bad-input-lines** switch. Each bad line has the source file name and line number prepended to it, separated from each other and the source line by colons ('**:**').

Field Constraints

Due to the way SiLK Flow records are stored, certain field combinations cannot be supported, certain fields must appear together, and some fields may only be used on certain occasions:

- Only two of the three time-related values (start time, duration, end time) may be specified. When all three are specified, the end time is ignored. This affects the `sTime,9`, `duration,10`, and `eTime,11` fields and the **--stime**, **--duration**, and **--etime** switches.
- Both ICMP type and ICMP code must be present when one is present. These may be set by a combination of the `iType` and `iCode` fields and the **--icmp-type** and **--icmp-code** switches. These values are ignored unless either the protocol is ICMP (1) or the record contains IPv6 addresses and the protocol is ICMPv6 (58). The ICMP type and code are encoded in the destination port field (`dPort,4` or **--dport**), and they overwrite the port value for ICMP and ICMPv6 flow records.
- Both initial TCP flags and session TCP flags must be present when one is present. These may be set by a combination of the `initialFlags,26` and `sessionFlags,27` fields and the **--flags-initial** and **--flags-session** switches. These fields are set to 0 for non-TCP flow records. When either field has a non-zero value, any value in the (ALL) TCP flags field (`flags,8` or **--flags-all**) is overwritten for TCP flow records.
- If the **silk.conf(5)** file defines more than one class, both class and type must be present for the values to have any affect on the SiLK flow record. These may be set by a combination of the `class` and `type` fields and the **--class** and **--type** switches. If *silk.conf* defines a single class, that class is used by default. The class and type must map to a valid pair; use **rwsiteinfo --fields=class,type** to see the list of valid class/type pairs for your site (cf. **rwsiteinfo(1)**).

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--fields=*FIELDS*

FIELDS contains the list of fields (columns) to parse. *FIELDS* is a comma separated list of field-names, field-integers, and ranges of field-integers; a range is specified by separating the start and end of the range with a hyphen (**-**). Field-names are case insensitive. A field name may not be specified more than once. (As of SiLK 3.15.0, `ignore` may appear multiple times, allowing multiple input fields to be ignored.)

A field is ignored when its name corresponds to a fixed value switch (e.g. **--protocol**) given on the command line (see Fixed Values).

The field names and their descriptions are:

ignore
: a field that **rwtuc** is to skip

sIP,1
: source IP address in the canonical form: dotted-quad for IPv4 or hex-encoded for IPv6 (when SiLK has been compiled with IPv6 support). Integers from 0 to 4294967295 are treated as IPv4 addresses.

dIP,2
: destination IP address in the same format as `sIP,1`

sPort,3
: source port as an integer from 0 to 65535 inclusive

dPort,4
: destination port as an integer from 0 to 65535 inclusive (cf. Field Constraints)

protocol,5
: IP protocol as an integer from 0 to 255 inclusive

packets,pkts,6
: packet count as an integer from 1 to 4294967295 inclusive

bytes,7
: byte count as an integer from 1 to 4294967295 inclusive

flags,8
: bit-wise OR of TCP flags over all packets in the flow; the string may contain `F`, `S`, `R`, `P`, `A`, `U`, `E`, `C` in upper- or lowercase (cf. Field Constraints)

sTime,9
: starting time of the flow, in the form `YYYY/MM/DD[:hh[:mm[:ss[.sss]]]]`. The letter `T` may be used in place of `:` to separate the day and hour fields. A floating point value between 536870912 and 4294967295 is also allowed and is treated as seconds since the UNIX epoch.

duration,10
: duration of flow as a floating point value from 0.0 to 4294967.295

eTime,11
: end time of flow in the same form as `sTime,9` (cf. Field Constraints)

sensor,12
: router sensor name or ID as given in *silk.conf* (cf. **silk.conf(5)**)

class
: class of router at collection point as given in *silk.conf* (cf. Field Constraints)

type
: type of router at collection point as given in *silk.conf* (cf. Field Constraints)

in,13
: router SNMP input interface or vlanId; an integer from 0 to 65535

out,14
: router SNMP output interface or postVlanId; an integer from 0 to 65535

nhIP,15

router next hop IP address in the same format as `sIP,1`

initialFlags,26

TCP flags on first packet in the flow; same form as the `flags,8` field (cf. Field Constraints)

sessionFlags,27

bit-wise OR of TCP flags on the second through final packet in the flow; same form as the `flags,8` field (cf. Field Constraints)

attribute,28

flow attributes set by the flow generator:

`S`

all the packets in this flow record are exactly the same size

`F`

flow generator saw additional packets in this flow following a packet with a FIN flag (excluding ACK packets)

`T`

flow generator prematurely created a record for a long-running connection due to a timeout. (When the flow generator **yaf(1)** is run with the **--silk** switch, it prematurely creates a flow and mark it with `T` if the byte count of the flow cannot be stored in a 32-bit value.)

`C`

flow generator created this flow as a continuation of long-running connection, where the previous flow for this connection met a timeout (or a byte threshold in the case of **yaf**).

Consider a long-running ssh session that exceeds the flow generator's *active* timeout. (This is the active timeout since the flow generator creates a flow for a connection that still has activity). The flow generator will create multiple flow records for this ssh session, each spanning some portion of the total session. The first flow record will be marked with a `T` indicating that it hit the timeout. The second through next-to-last records will be marked with `TC` indicating that this flow both timed out and is a continuation of a flow that timed out. The final flow will be marked with a `C`, indicating that it was created as a continuation of an active flow.

application,29

guess as to the content of the flow, as an integer from 0 to 65535. Some software that generates flow records from packet data, such as **yaf**, will inspect the contents of the packets that make up a flow and use traffic signatures to label the content of the flow. SiLK calls this label the *application*; **yaf** refers to it as the *appLabel*. The application is the port number that is traditionally used for that type of traffic (see the */etc/services* file on most UNIX systems). For example, traffic that the flow generator recognizes as FTP will have a value of 21, even if that traffic is being routed through the standard HTTP/web port (80).

iType

ICMP type as an integer from 0 to 255 inclusive (cf. Field Constraints)

iCode

ICMP code as an integer from 0 to 255 inclusive (cf. Field Constraints)

--column-separator=*CHAR*

Use the character *CHAR* as the delimiter between columns (fields) in the input. The default column separator is the vertical pipe ('|'). **rwtuc** normally ignores whitespace (space and tab) around the column separator; however, using space or tab as the separator causes *each* space or tab character to be treated as a field delimiter. The newline character is not a valid delimiter character since it is used to denote records.

--output-path=*PATH*

Write the binary SiLK Flow records to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output. If *PATH* names an existing file, **rwtuc** exits with an error unless the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. When *PATH* ends in `.gz`, the output is compressed using the library associated with **gzip(1)**. If this switch is not given, the output is written to the standard output. Attempting to write the binary output to a terminal causes **rwtuc** to exit with an error.

--bad-input-lines=*FILEPATH*

Copy any lines which could not be parsed to *FILEPATH*. The strings `stdout` and `stderr` may be used for the standard output and standard error, respectively. Each bad line is prepended by the source input file, a colon, the line number, and a colon. On exit, **rwtuc** removes *FILEPATH* if all input lines were successfully parsed.

--verbose

When an input line fails to parse, print a message to the standard error describing the problem. When this switch is not specified, parsing failures are not reported. **rwtuc** continues to process the input after printing the message. To stop processing when a parsing error occurs, use **--stop-on-error**.

--stop-on-error

When an input line fails to parse, print a message to the standard error describing the problem and exit the program. When this occurs, the output file contains any records successfully created prior to reading the bad input line. The default behavior of **rwtuc** is to silently ignore parsing errors. To report parsing errors and continue processing the input, use **--verbose**.

--no-titles

Parse the first line of the input as field values. Normally when the **--fields** switch is specified, **rwtuc** examines the first line to determine if the line contains the names (titles) of fields and skips the line if it does. **rwtuc** exits with an error when **--no-titles** is given but **--fields** is not.

--note-add=*TEXT*

Add the specified *TEXT* to the header of the output file as an annotation. This switch may be repeated to add multiple annotations to a file. To view the annotations, use the **rwfileinfo(1)** tool.

--note-file-add=*FILENAME*

Open *FILENAME* and add the contents of that file to the header of the output file as an annotation. This switch may be repeated to add multiple annotations. Currently the application makes no effort to ensure that *FILENAME* contains text; be careful that you do not attempt to add a SiLK data file as an annotation.

--compression-method=*COMP_METHOD*

Specify the compression library to use when writing output files. If this switch is not given, the value in the SILK_COMPRESSION_METHOD environment variable is used if the value names an available compression method. When no compression method is specified, output to the standard output or to named pipes is not compressed, and output to files is compressed using the default chosen when SiLK was compiled. The valid values for *COMP_METHOD* are determined by which external libraries were found when SiLK was compiled. To see the available compression methods and the default method, use the **--help** or **--version** switch. SiLK can support the following *COMP_METHOD* values when the required libraries are available.

none

Do not compress the output using an external library.

zlib

Use the **zlib(3)** library for compressing the output, and always compress the output regardless of the destination. Using zlib produces the smallest output files at the cost of speed.

lzo1x

Use the *lzo1x* algorithm from the LZO real time compression library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead.

snappy

Use the *snappy* library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead. *Since SiLK 3.13.0.*

best

Use lzo1x if available, otherwise use snappy if available, otherwise use zlib if available. Only compress the output when writing to a file.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwtuc** searches for the site configuration file in the locations specified in the FILES section.

--xargs

--xargs=*FILENAME*

Read the names of the input files from *FILENAME* or from the standard input if *FILENAME* is not provided. The input is expected to have one filename per line. **rwtuc** opens each named file in turn and reads text from it as if the filenames had been listed on the command line. *Since SiLK 3.15.0.*

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

Fixed Values

The following switches may be used to set fields to fixed values. A value specified using one these switches overrides the field when it appears in the input, causing that column of input to be completely ignored.

--saddress=*IPADDR*

Set the source address field to *IPADDR* for all records. *IPADDR* may be in canonical notation or an unsigned integer.

--daddress=*IPADDR*

Set the destination address field to *IPADDR* for all records. *IPADDR* may be in canonical notation or an unsigned integer.

--sport=*NUM*

Set the source port field to *NUM* for all records; a value between 0 and 65535.

--dport=*NUM*

Set the destination port field to *NUM* for all records; a value between 0 and 65535. (cf. Field Constraints)

--protocol=*NUM*

Set the protocol field to *NUM* for all records; a value between 0 and 255.

--packets=*NUM*

Set the packets field to *NUM* for all records; the value must be non-zero.

--bytes=*NUM*

Set the bytes field to *NUM* for all records; the value must be non-zero.

--flags-all=*TCPFLAGS*

Set the TCP flags field to *TCPFLAGS* for all records. (cf. Field Constraints)

--stime=*TIME*

Set the start time field to *TIME* for all records.

--duration=*NUM*

Set the duration field to *NUM* for all records.

--etime=*TIME*

Set the end time field to *TIME* for all records. (cf. Field Constraints)

--sensor=*SID*

Set the sensor field to *SID* for all records. This may either be a sensor name or sensor ID.

--input-index=*NUM*

Set the SNMP input index field to *NUM* for all records; a value between 0 and 65535.

--output-index=*NUM*

Set the SNMP output index field to *NUM* for all records; a value between 0 and 65535.

--next-hop-ip=*IPADDR*

Set the next-hop-ip field to *IPADDR* for all records. *IPADDR* may be in canonical notation or an unsigned integer.

--flags-initial=*TCPFLAGS*

Set the initial TCP flags field to *TCPFLAGS* for all records. (cf. Field Constraints)

--flags-session=*TCPFLAGS*

Set the session TCP flags field to *TCPFLAGS* for all records. (cf. Field Constraints)

--attributes=*ATTR*

Set the attributes field to *ATTR* for all records.

--application=*NUM*

Set the application field to *NUM* for all records; a value between 0 and 65535.

--class=*NAME*

Set the class field to *NAME* for all records. (cf. Field Constraints)

--type=*NAME*

Set the type field to *NAME* for all records. (cf. Field Constraints)

--icmp-type=*NUM*

Set the ICMP type field to *NUM* for all ICMP or ICMPv6 flow records; a value between 0 and 255. (cf. Field Constraints)

--icmp-code=*NUM*

Set the ICMP code field to *NUM* for all ICMP or ICMPv6 flow records; a value between 0 and 255. (cf. Field Constraints)

--stime+msec=*TIME*

An alias for **--stime**. This switch is deprecated as of SiLK 3.6.0, and it will be removed in the SiLK 4.0 release.

--etime+msec=*TIME*

An alias for **--etime**. This switch is deprecated as of SiLK 3.6.0, and it will be removed in the SiLK 4.0 release.

--duration+msec=*NUM*

An alias for **--duration**. This is is deprecated as of SiLK 3.6.0, and it will be removed in the SiLK 4.0 release.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

Using **rwtuc** to parse the output of **rwcut(1)** should produce the same output:

```
$ rwcut data.rw > cut.txt
$ md5 < cut.txt
7e3d693cd2cba2510803935274e1debd
$ rwtuc < cut.txt | rwcut | md5
7e3d693cd2cba2510803935274e1debd
```

To swap the source IP and port with the destination IP and port in *flows.rw* and save the result in *reverse.rw*:

```
$ rwcut --fields=dip,dport,sip,sport,5-15,20-29 flows.rw   \
  | rwtuc --fields=1-15,20-29 --output-path=reverse.rw
```

rwtuc may be used to obfuscate the flow data in *myflows.rw* to produce *obflows.rw*. Pipe the output from **rwcut** into a script that manipulates the IP addresses, then pipe that into **rwtuc**. Using the **sed(1)** script in *priv.sed*, the invocation is:

```
$ rwcut --fields=1-10,13-15,26-29 myflows.rw               \
  | sed -f priv.sed                                        \
  | rwtuc --sensor=1 > obflows.rw
```

If the first line of input appears to contain titles, **rwtuc** ignores it. In the first invocation below, **rwtuc** treats `SP` as an abbreviation for `sPort` and ignores the line. Use the **--no-titles** switch to force **rwtuc** to parse the line:

```
$ echo 'SP' | rwtuc --fields=flags | rwcut --fields=flags
   flags|
$
$ echo 'SP' | rwtuc --fields=flags --no-titles | rwcut --fields=flags
   flags|
 S P    |
$
```

By default, **rwtuc** silently ignores lines that it cannot parse. Use the **--verbose** flag to see error messages:

```
$ echo sport | rwtuc --fields=flags --no-titles --verbose >/dev/null
rwtuc: stdin:1: Invalid flags 'sport': Unexpected character 'o'
```

ENVIRONMENT

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_COMPRESSION_METHOD

This environment variable is used as the value for **--compression-method** when that switch is not provided. *Since SiLK 3.13.0.*

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwtuc** may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **rwtuc** may use this environment variable. See the FILES section for details.

TZ

When a SiLK installation is built to use the local timezone (to determine if this is the case, check the `Timezone support` value in the output from **rwtuc --version**), the value of the TZ environment variable determines the timezone in which **rwtuc** parses timestamps. If the TZ environment variable is not set, the default timezone is used. Setting TZ to 0 or the empty string causes timestamps to be parsed as UTC. The value of the TZ environment variable is ignored when the SiLK installation uses utc. For system information on the TZ variable, see **tzset(3)** or **environ(7)**.

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

SEE ALSO

rwcut(1), **rwfileinfo(1)**, **rwsiteinfo(1)**, **silk.conf(5)**, **silk(7)**, **yaf(1)**, **gzip(1)**, **sed(1)**, **zlib(3)**, **tzset(3)**, **environ(7)**

rwuniq

Bin SiLK Flow records by a key and print each bin's volume

SYNOPSIS

```
rwuniq --fields=KEY [--values=VALUES]
      [{--threshold=MIN-MAX | --threshold=MIN}]
      [--presorted-input] [--sort-output]
      [{--bin-time=SECONDS | --bin-time}]
      [--timestamp-format=FORMAT] [--epoch-time]
      [--ip-format=FORMAT] [--integer-ips] [--zero-pad-ips]
      [--integer-sensors] [--integer-tcp-flags]
      [--no-titles] [--no-columns] [--column-separator=CHAR]
      [--no-final-delimiter] [{--delimited | --delimited=CHAR}]
      [--print-filenames] [--copy-input=PATH] [--output-path=PATH]
      [--pager=PAGER_PROG] [--temp-directory=DIR_PATH]
      [{--legacy-timestamps | --legacy-timestamps={1,0}}]
      [--all-counts] [{--bytes | --bytes=MIN | --bytes=MIN-MAX}]
      [{--packets | --packets=MIN | --packets=MIN-MAX}]
      [{--flows | --flows=MIN | --flows=MIN-MAX}]
      [--stime] [--etime]
      [{--sip-distinct | --sip-distinct=MIN | --sip-distinct=MIN-MAX}]
      [{--dip-distinct | --dip-distinct=MIN | --dip-distinct=MIN-MAX}]
      [--ipv6-policy={ignore,asv4,mix,force,only}]
      [--site-config-file=FILENAME]
      [--plugin=PLUGIN [--plugin=PLUGIN ...]]
      [--python-file=PATH [--python-file=PATH ...]]
      [--pmap-file=MAPNAME:PATH [--pmap-file=MAPNAME:PATH ...]]
      [--pmap-column-width=NUM]
      {[--xargs] | [--xargs=FILENAME] | [FILE [FILE ...]]}

rwuniq [--pmap-file=MAPNAME:PATH [--pmap-file=MAPNAME:PATH ...]]
      [--plugin=PLUGIN ...] [--python-file=PATH ...] --help

rwuniq [--pmap-file=MAPNAME:PATH [--pmap-file=MAPNAME:PATH ...]]
      [--plugin=PLUGIN ...] [--python-file=PATH ...] --help-fields

rwuniq --version
```

DESCRIPTION

rwuniq reads SiLK Flow records and groups them by a key composed of user-specified attributes of the flows. For each group (or bin), a collection of user-specified *aggregate values* is computed; these values are typically related to the volume of the bin, such as the sum of the bytes fields for all records that match the key. Once all the SiLK Flow records are read, the key fields and the aggregate values are printed. For some of the built-in aggregate values, it is possible to limit the output to the bins where the aggregate value meets a user-specified minimum and/or maximum.

There is no need to sort the input to **rwuniq** since **rwuniq** normally rearranges the records as they are read. To have **rwuniq** sort its output, use the **--sort-output** switch.

rwuniq reads SiLK Flow records from the files named on the command line or from the standard input when no file names are specified and **--xargs** is not present. To read the standard input in addition to the named files, use `-` or `stdin` as a file name. If an input file name ends in `.gz`, the file is uncompressed as it is read. When the **--xargs** switch is provided, **rwuniq** reads the names of the files to process from the named text file or from the standard input if no file name argument is provided to the switch. The input to **--xargs** must contain one file name per line.

The user must provide the **--fields** switch to select the flow attribute(s) (or field(s)) that comprise the key for each bin. The available fields are similar to those supported by **rwcut(1)**; see the description of the **--fields** switch in the OPTIONS section below for the details. The list of fields can be extended by loading PySiLK files (see **silkpython(3)**) or plug-ins (**silk-plugin(3)**). The fields are printed in the order in which they occur in the **--fields** switch. The size of the key is limited to 256 octets. A larger key more quickly uses the available the memory leading to slower performance.

The aggregate value(s) to compute for each bin are also chosen by the user. As with the key fields, the user can extend the list of aggregate fields by using PySiLK or plug-ins. Specify the aggregate fields with the **--values** switch; the aggregate fields are printed in the order they occur in the **--values** switch. If the user does not provide **--values** or a **--threshold** switch (described next), **rwuniq** defaults to computing the number of flow records for each bin. As with the key fields, requesting more aggregate values slows performance.

The **--threshold** switch (added in SiLK 3.17.0) allows the user to print only bins where a value field is within a certain range. The switch's argument contains the name of the value field, an equals sign, the minimum value (start of the range), and optionally a hyphen and the maximum value (end of the range); e.g., `--threshold=bytes=1000-2000`. The upper bound is unlimited when no maximum is specified. The **--threshold** switch may be repeated to set multiple thresholds, and only those bins that meet all thresholds are printed. Each field named by **--threshold** is appended to the set of aggregate value fields unless that field was named in the **--values** switch.

The **--presorted-input** switch may allow **rwuniq** to process data more efficiently by causing **rwuniq** to assume the input has been previously sorted with the **rwsort(1)** command. With this switch, **rwuniq** typically does not need large amounts of memory because it does not bin each flow; instead, it keeps a running summation and outputs the bin whenever the key changes. For the output to be meaningful, **rwsort** and **rwuniq** *must* be invoked with the same **--fields** value. When multiple input files are specified and **--presorted-input** is given, **rwuniq** merge-sorts the flow records from the input files. **rwuniq** typically runs faster if you do *not* include the **--presorted-input** switch when counting distinct values, even when reading sorted input. Finally, you may get unusual results with **--presorted-input** when the **--fields** switch contains multiple time-related key fields (`sTime`, `duration`, `eTime`), or when the time-related key is not the final key listed in **--fields**; see the NOTES section for details.

rwuniq attempts to keep all key and aggregate value data in the computer's memory. If **rwuniq** runs out of memory, the current key and aggregate value data is written to a temporary file. Once all input has been processed, the data from the temporary files is merged to produce the final output. By default, these temporary files are stored in the */tmp* directory. Because these files can be large, it is strongly recommended that */tmp not* be used as the temporary directory. To modify the temporary directory used by **rwuniq**, provide the **--temp-directory** switch, set the SILK_TMPDIR environment variable, or set the TMPDIR environment variable.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

The **--fields** switch is required. **rwuniq** fails when it is not provided.

--fields=*KEY*

KEY contains the list of flow attributes (a.k.a. fields or columns) that make up the key into which flows are binned. The columns are displayed in the order the fields are specified. Each field may be specified once only. *KEY* is a comma separated list of field-names, field-integers, and ranges of field-integers; a range is specified by separating the start and end of the range with a hyphen (**-**). Field-names are case insensitive. Example:

```
--fields=stime,10,1-5
```

There is no default value for the **--fields** switch; the switch must be specified.

The complete list of built-in fields that the SiLK tool suite supports follows, though note that not all fields are present in all SiLK file formats; when a field is not present, its value is 0.

sIP,1
: source IP address

dIP,2
: destination IP address

sPort,3
: source port for TCP and UDP, or equivalent

dPort,4
: destination port for TCP and UDP, or equivalent. See note at `iType`.

protocol,5
: IP protocol

packets,pkts,6
: packet count

bytes,7
: byte count

flags,8
: bit-wise OR of TCP flags over all packets

sTime,9
: starting time of flow (seconds resolution unless **--bin-time** includes fractional seconds). When the time-related fields `sTime`,`duration`,`eTime` are all in use, **rwuniq** ignores the final time field when binning the records.

duration,10
: duration of flow (seconds resolution unless **--bin-time** includes fractional seconds). This field is not adjusted by **--bin-time** unless **--fields** includes both `sTime` and `eTime`. See note at `sTime,9`.

eTime,11
: end time of flow (seconds resolution unless **--bin-time** includes fractional seconds). See note at `sTime,9`.

sensor,12
: name or ID of the sensor where the flow was collected

class,20
: class assigned to the flow by **rwflowpack(8)**. Binning by `class` and/or `type` equates to binning by the integer value used internally to represent the class/type pair. When **--fields** contains `class` but not `type`, **rwuniq**'s output contains multiple rows with the same value(s) for the key field(s).

type,21
: type assigned to the flow by **rwflowpack(8)**. See note on previous entry.

iType
: the ICMP type value for ICMP or ICMPv6 flows and empty (numerically zero) for non-ICMP flows. Internally, SiLK stores the ICMP type and code in the `dPort` field. To avoid getting very odd results, either do not use the `dPort` field when your key includes ICMP field(s) or be certain to include the `protocol` field as part of your key. This field was introduced in SiLK 3.8.1.

iCode
: the ICMP code value for ICMP or ICMPv6 flows and empty for non-ICMP flows. See note at `iType`.

icmpTypeCode,25
: equivalent to `iType`,`iCode` when used in **--fields**. This field may not be mixed with `iType` or `iCode`, and this field is deprecated as of SiLK 3.8.1. As of SiLK 3.8.1, `icmpTypeCode` may no longer be used as the argument to the `Distinct:` value field; the `dPort` field provides an equivalent result as long as the input is limited to ICMP flow records.

Many SiLK file formats do not store the following fields and their values are always be 0; they are listed here for completeness:

in,13
: router SNMP input interface or vlanId if packing tools were configured to capture it (see **sensor.conf(5)**)

out,14
: router SNMP output interface or postVlanId

nhIP,15
: router next hop IP

SiLK can store flows generated by enhanced collection software that provides more information than NetFlow v5. These flows may support some or all of these additional fields; for flows without this additional information, the field's value is always 0.

initialFlags,26
: TCP flags on first packet in the flow

sessionFlags,27
: bit-wise OR of TCP flags over all packets except the first in the flow

attributes,28
: flow attributes set by the flow generator:

  `S`
  : all the packets in this flow record are exactly the same size

`F`
: flow generator saw additional packets in this flow following a packet with a FIN flag (excluding ACK packets)

`T`
: flow generator prematurely created a record for a long-running connection due to a timeout. (When the flow generator **yaf(1)** is run with the **--silk** switch, it prematurely creates a flow and mark it with `T` if the byte count of the flow cannot be stored in a 32-bit value.)

`C`
: flow generator created this flow as a continuation of long-running connection, where the previous flow for this connection met a timeout (or a byte threshold in the case of **yaf**).

Consider a long-running ssh session that exceeds the flow generator's *active* timeout. (This is the active timeout since the flow generator creates a flow for a connection that still has activity). The flow generator will create multiple flow records for this ssh session, each spanning some portion of the total session. The first flow record will be marked with a `T` indicating that it hit the timeout. The second through next-to-last records will be marked with `TC` indicating that this flow both timed out and is a continuation of a flow that timed out. The final flow will be marked with a `C`, indicating that it was created as a continuation of an active flow.

application,29
: guess as to the content of the flow. Some software that generates flow records from packet data, such as **yaf**, will inspect the contents of the packets that make up a flow and use traffic signatures to label the content of the flow. SiLK calls this label the *application*; **yaf** refers to it as the *appLabel*. The application is the port number that is traditionally used for that type of traffic (see the */etc/services* file on most UNIX systems). For example, traffic that the flow generator recognizes as FTP will have a value of 21, even if that traffic is being routed through the standard HTTP/web port (80).

The following fields provide a way to label the IPs or ports on a record. These fields require external files to provide the mapping from the IP or port to the label:

sType,16
: for the source IP address, the value 0 if the address is non-routable, 1 if it is internal, or 2 if it is routable and external. Uses the mapping file specified by the SILK_ADDRESS_TYPES environment variable, or the *address_types.pmap* mapping file, as described in **addrtype(3)**.

dType,17
: as **sType** for the destination IP address

scc,18
: for the source IP address, a two-letter country code abbreviation denoting the country where that IP address is located. Uses the mapping file specified by the SILK_COUNTRY_CODES environment variable, or the *country_codes.pmap* mapping file, as described in **ccfilter(3)**. The abbreviations are those defined by ISO 3166-1 (see for example https://www.iso.org/iso-3166-country-codes.html or https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2) or the following special codes: **--** N/A (e.g. private and experimental reserved addresses); **a1** anonymous proxy; **a2** satellite provider; **o1** other

dcc,19
: as **scc** for the destination IP

src-*map-name*
: label contained in the prefix map file associated with *map-name*. If the prefix map is for IP addresses, the label is that associated with the source IP address. If the prefix map is for protocol/port pairs, the label is that associated with the protocol and source port. See also the description of the **--pmap-file** switch below and the **pmapfilter(3)** manual page.

dst-*map-name*

as **src-*map-name*** for the destination IP address or the protocol and destination port.

sval

as **src-*map-name*** when no map-name is associated with the prefix map file

dval

as **dst-*map-name*** when no map-name is associated with the prefix map file

Finally, the list of built-in fields may be augmented by the run-time loading of PySiLK code or plug-ins written in C (also called shared object files or dynamic libraries), as described by the **--python-file** and **--plugin** switches.

--values=*VALUES*

Specify the aggregate values to compute for each bin as a comma separated list of names. Names are case insensitive. When the **--threshold** switch specifies an aggregate value field that does appear in *VALUES*, that field is appended to *VALUES*. When neither the **--values** switch nor any **--threshold** switch is specified, **rwuniq** counts the number of flow records for each bin. The aggregate fields are printed in the order they occur in *VALUES*. The names of the built-in value fields follow. This list can be augmented through the use of PySiLK and plug-ins.

Records

Count the number of flow records that mapped to each bin.

Packets

Sum the number of packets across all records that mapped to each bin.

Bytes

Sum the number of bytes across all records that mapped to each bin.

sTime-Earliest

Keep track of the earliest start time (minimum time) seen across all records that mapped to each bin, in seconds resolution. The **--bin-time** switch does not normally affect this value; however, this value uses milliseconds resolution when **--bin-time** includes fractional seconds.

eTime-Latest

Keep track of the latest end time (maximum time) seen across all records that mapped to each bin, in seconds resolution. The **--bin-time** switch does not normally affect this value; however, this value uses milliseconds resolution when **--bin-time** includes fractional seconds.

sIP-Distinct

Count the number of distinct source IP addresses that were seen for each bin, an alias for Distinct:sIP.

dIP-Distinct

Count the number of distinct destination IP addresses that were seen for each bin, an alias for Distinct:dIP.

Distinct:*KEY_FIELD*

Count the number of distinct values for *KEY_FIELD*, where *KEY_FIELD* is any field that can be used as an argument to **--fields** except `icmpTypeCode`. For example, `Distinct:sPort` counts the number of distinct source ports for each bin. When this aggregate value field is used, the specified *KEY_FIELD* cannot be present in the argument to **--fields**.

Flows

Count the number of flow records that mapped to each bin; an alias for Records.

--plugin=*PLUGIN*

Augment the list of key fields and/or aggregate value fields by using run-time loading of the plug-in (shared object) whose path is *PLUGIN*. The switch may be repeated to load multiple plug-ins. The creation of plug-ins is described in the **silk-plugin(3)** manual page. When *PLUGIN* does not contain a slash (/), **rwuniq** attempts to find a file named *PLUGIN* in the directories listed in the FILES section. If **rwuniq** finds the file, it uses that path. If *PLUGIN* contains a slash or if **rwuniq** does not find the file, **rwuniq** relies on your operating system's **dlopen(3)** call to find the file. When the SILK_PLUGIN_DEBUG environment variable is non-empty, **rwuniq** prints status messages to the standard error as it attempts to find and open each of its plug-ins.

--threshold=*VALUE_FIELD*=*MIN*-*MAX*

--threshold=*VALUE_FIELD*=*MIN*

Limit the output of **rwuniq** to the bins where the value of the aggregate value field *VALUE_FIELD* is not less than *MIN* and not more than *MAX*. If *MAX* is not given, limit the output to the bins where the value of *VALUE_FIELD* is at least *MIN*. The *VALUE_FIELD* argument is case insensitive and may be abbreviated to the shortest unique prefix. This switch may be repeated to set thresholds for multiple fields, and **rwuniq** only prints bins that meet all thresholds. A *MIN* of 0 is treated as 1. If *VALUE_FIELD* is not present in the argument to the **--values** switch, it is appended to those aggregate values. *VALUE_FIELD* may be **Records** (or **Flows)**, **Packets**, **Bytes**, **sIP-Distinct**, **dIP-Distinct**, or **Distinct:***KEY_FIELD*. Setting thresholds for aggregate value fields defined by plug-ins is not supported. *Since SiLK 3.17.0.*

Miscellaneous options:

--presorted-input

Cause **rwuniq** to assume that it is reading sorted input; i.e., that **rwuniq**'s input file(s) were generated by **rwsort(1)** using the *exact same* value for the **--fields** switch. When no distinct counts are being computed, **rwuniq** can process its input without needing to write temporary files. When multiple input files are specified, **rwuniq** merge-sorts the flow records from the input files. See the NOTES section for issues that may occur when using **--presorted-input**.

--sort-output

Cause **rwuniq** to present the output in sorted numerical order. The key **rwuniq** uses for sorting is the same key it uses to index each bin.

--bin-time=*SECONDS*

--bin-time

Adjust the times in the key fields `sTime` and `eTime` to appear on *SECONDS*-second boundaries (the floor of the time is used). As of SiLK 3.17.0, *SECONDS* may be a fractional value of 0.001 or greater, and **rwuniq** uses millisecond timestamps when *SECONDS* includes a fractional value that is non-zero. When this switch is not specified, times appear on 1-second boundaries. When the switch is used but no argument is given, **rwuniq** uses 60-second time bins. (When the start-time is the only key field and time binning is desired, consider using **rwcount(1)** instead.)

--timestamp-format=*FORMAT*

Specify the format and/or timezone to use when printing timestamps. When this switch is not specified, the SILK_TIMESTAMP_FORMAT environment variable is checked for a default format and/or timezone. If it is empty or contains invalid values, timestamps are printed in the default format, and the timezone is UTC unless SiLK was compiled with local timezone support. *FORMAT* is a comma-separated list of a format and/or a timezone. The format is one of:

default
Print the timestamps as *YYYY*/*MM*/*DD*T*hh*:*mm*:*ss*.

iso
Print the timestamps as *YYYY*-*MM*-*DD hh*:*mm*:*ss*.

m/d/y
Print the timestamps as *MM*/*DD*/*YYYY hh*:*mm*:*ss*.

epoch
Print the timestamps as the number of seconds since 00:00:00 UTC on 1970-01-01.

When a timezone is specified, it is used regardless of the default timezone support compiled into SiLK. The timezone is one of:

utc
Use Coordinated Universal Time to print timestamps.

local
Use the TZ environment variable or the local timezone.

--epoch-time
Print timestamps as epoch time (number of seconds since midnight GMT on 1970-01-01). This switch is equivalent to **--timestamp-format=epoch**, it is deprecated as of SiLK 3.0.0, and it will be removed in the SiLK 4.0 release.

--ip-format=*FORMAT*
Specify how IP addresses are printed, where *FORMAT* is a comma-separated list of the arguments described below. When this switch is not specified, the SILK_IP_FORMAT environment variable is checked for a value and that format is used if it is valid. The default *FORMAT* is `canonical`. *Since SiLK 3.7.0.*

canonical
Print IP addresses in the canonical format. If the key only contains IPv4 addresses, use dot-separated decimal (`192.0.2.1`). Otherwise, use colon-separated hexadecimal (`2001:db8::1`) or a mixed IPv4-IPv6 representation for IPv4-mapped IPv6 addresses (the ::ffff:0:0/96 netblock, e.g., `::ffff:192.0.2.1`) and IPv4-compatible IPv6 addresses (the ::/96 netblock other than ::/127, e.g., `::192.0.2.1`).

no-mixed
Print IP addresses in the canonical format (`192.0.2.1` or `2001:db8::1`) but do not used the mixed IPv4-IPv6 representations. For example, use `::ffff:c000:201` instead of `::ffff:192.0.2.1`. *Since SiLK 3.17.0.*

decimal
Print IP addresses as integers in decimal format. For example, print `192.0.2.1` and `2001:db8::1` as `3221225985` and `42540766411282592856903984951653826561`, respectively.

hexadecimal
Print IP addresses as integers in hexadecimal format. For example, print `192.0.2.1` and `2001:db8::1` as `c00000201` and `20010db80000000000000000000000001`, respectively.

zero-padded
Make all IP address strings contain the same number of characters by padding numbers with leading zeros. For example, print `192.0.2.1` and `2001:db8::1` as `192.000.002.001` and `2001:0db8:0000:0000:0000:0000:0000:0001`, respectively. For

IPv6 addresses, this setting implies `no-mixed`, so that `::ffff:192.0.2.1` is printed as `0000:0000:0000:0000:0000:ffff:c000:0201`. As of SiLK 3.17.0, may be combined with any of the above, including `decimal` and `hexadecimal`.

The following arguments modify certain IP addresses prior to printing. These arguments may be combined with the above formats.

map-v4
Change IPv4 addresses to IPv4-mapped IPv6 addresses (addresses in the ::ffff:0:0/96 netblock) prior to formatting. *Since SiLK 3.17.0.*

unmap-v6
When the key contains IPv6 addresses, change any IPv4-mapped IPv6 addresses (addresses in the ::ffff:0:0/96 netblock) to IPv4 addresses prior to formatting. *Since SiLK 3.17.0.*

The following argument is also available:

force-ipv6
Set *FORMAT* to `map-v4,no-mixed`.

--integer-ips
Print IP addresses as integers. This switch is equivalent to **--ip-format=decimal**, it is deprecated as of SiLK 3.7.0, and it will be removed in the SiLK 4.0 release.

--zero-pad-ips
Print IP addresses as fully-expanded, zero-padded values in their canonical form. This switch is equivalent to **--ip-format=zero-padded**, it is deprecated as of SiLK 3.7.0, and it will be removed in the SiLK 4.0 release.

--integer-sensors
Print the integer ID of the sensor rather than its name.

--integer-tcp-flags
Print the TCP flag fields (flags, initialFlags, sessionFlags) as an integer value. Typically, the characters `F,S,R,P,A,U,E,C` are used to represent the TCP flags.

--no-titles
Turn off column titles. By default, titles are printed.

--no-columns
Disable fixed-width columnar output.

--column-separator=*C*
Use specified character between columns and after the final column. When this switch is not specified, the default of '|' is used.

--no-final-delimiter
Do not print the column separator after the final column. Normally a delimiter is printed.

--delimited

--delimited=*C*
Run as if **--no-columns --no-final-delimiter --column-sep**=*C* had been specified. That is, disable fixed-width columnar output; if character *C* is provided, it is used as the delimiter between columns instead of the default '|'.

--print-filenames

Print to the standard error the names of input files as they are opened.

--copy-input=*PATH*

Copy all binary SiLK Flow records read as input to the specified file or named pipe. *PATH* may be `stdout` or `-` to write flows to the standard output as long as the **--output-path** switch is specified to redirect **rwuniq**'s textual output to a different location.

--output-path=*PATH*

Write the textual output to *PATH*, where *PATH* is a filename, a named pipe, the keyword `stderr` to write the output to the standard error, or the keyword `stdout` or `-` to write the output to the standard output (and bypass the paging program). If *PATH* names an existing file, **rwuniq** exits with an error unless the SILK_CLOBBER environment variable is set, in which case *PATH* is overwritten. If this switch is not given, the output is either sent to the pager or written to the standard output.

--pager=*PAGER_PROG*

When output is to a terminal, invoke the program *PAGER_PROG* to view the output one screen full at a time. This switch overrides the SILK_PAGER environment variable, which in turn overrides the PAGER variable. If the **--output-path** switch is given or if the value of the pager is determined to be the empty string, no paging is performed and all output is written to the terminal.

--ipv6-policy=*POLICY*

Determine how IPv4 and IPv6 flows are handled when SiLK has been compiled with IPv6 support. When the switch is not provided, the SILK_IPV6_POLICY environment variable is checked for a policy. If it is also unset or contains an invalid policy, the *POLICY* is **mix**. When SiLK has not been compiled with IPv6 support, IPv6 flows are always ignored, regardless of the value passed to this switch or in the SILK_IPV6_POLICY variable. The supported values for *POLICY* are:

ignore

Ignore any flow record marked as IPv6, regardless of the IP addresses it contains.

asv4

Convert IPv6 flow records that contain addresses in the ::ffff:0:0/96 netblock (that is, IPv4-mapped IPv6 addresses) to IPv4 and ignore all other IPv6 flow records.

mix

Process the input as a mixture of IPv4 and IPv6 flow records. When an IP address is used as part of the key or value, this policy is equivalent to **force**.

force

Convert IPv4 flow records to IPv6, mapping the IPv4 addresses into the ::ffff:0:0/96 netblock.

only

Process only flow records that are marked as IPv6 and ignore IPv4 flow records in the input.

--temp-directory=*DIR_PATH*

Specify the name of the directory in which to store data files temporarily when the memory is not large enough to store all the bins and their aggregate values. This switch overrides the directory specified in the SILK_TMPDIR environment variable, which overrides the directory specified in the TMPDIR variable, which overrides the default, */tmp*.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwuniq** searches for the site configuration file in the locations specified in the FILES section.

--legacy-timestamps

--legacy-timestamps=*NUM*

When *NUM* is not specified or is 1, this switch is equivalent to **--timestamp-format=m/d/y**. Otherwise, the switch has no effect. This switch is deprecated as of SiLK 3.0.0, and it will be removed in the SiLK 4.0 release.

--xargs

--xargs=*FILENAME*

Read the names of the input files from *FILENAME* or from the standard input if *FILENAME* is not provided. The input is expected to have one filename per line. **rwuniq** opens each named file in turn and reads records from it as if the filenames had been listed on the command line.

--help

Print the available options and exit. Specifying switches that add new fields, values, or additional switches before **--help** allows the output to include descriptions of those fields or switches.

--help-fields

Print the description and alias(es) of each field and value and exit. Specifying switches that add new fields before **--help-fields** allows the output to include descriptions of those fields.

--version

Print the version number and information about how SiLK was configured, then exit the application.

--pmap-file=*PATH*

--pmap-file=*MAPNAME*:*PATH*

Load the prefix map file located at *PATH* and create fields named src-*map-name* and dst-*map-name* where *map-name* is either the *MAPNAME* part of the argument or the map-name specified when the file was created (see **rwpmapbuild(1)**). If no map-name is available, **rwuniq** names the fields `sval` and `dval`. Specify *PATH* as `-` or `stdin` to read from the standard input. The switch may be repeated to load multiple prefix map files, but each prefix map must use a unique map-name. The **--pmap-file** switch(es) must precede the **--fields** switch. See also **pmapfilter(3)**.

--pmap-column-width=*NUM*

When printing a label associated with a prefix map, this switch gives the maximum number of characters to use when displaying the textual value of the field.

--python-file=*PATH*

When the SiLK Python plug-in is used, **rwuniq** reads the Python code from the file *PATH* to define additional fields that can be used as part of the key or as an aggregate value. This file should call **register_field()** for each field it wishes to define. For details and examples, see the **silkpython(3)** and **pysilk(3)** manual pages.

Deprecated volume switches

These options add the named aggregate field(s) to **--values** if the field is not present. When an argument is specified, the switch is equivalent to a **--threshold** switch. Use of these switches is deprecated.

--all-counts

Append the following fields to the argument of the **--values** switch unless the field is already present: **Bytes**, **Packets**, **Records**, **sTime-Earliest**, and **eTime-Latest**. Deprecated since SiLK 2.0.0.

--bytes

Append **Bytes** to the argument of the **--values** switch unless it is already present. Deprecated since SiLK 2.0.0.

--bytes=*MIN*

Add **--threshold=bytes**=*MIN* to the options. Deprecated since SiLK 3.17.0.

--bytes=*MIN-MAX*

Add **--threshold=bytes**=*MIN-MAX* to the options. Deprecated since SiLK 3.17.0.

--packets

Append **Packets** to the argument of the **--values** switch unless it is already present. Deprecated since SiLK 2.0.0.

--packets=*MIN*

Add **--threshold=packets**=*MIN* to the options. Deprecated since SiLK 3.17.0.

--packets=*MIN-MAX*

Add **--threshold=packets**=*MIN-MAX* to the options. Deprecated since SiLK 3.17.0.

--flows

Append **Records** to the argument of the **--values** switch unless it is already present. Deprecated since SiLK 2.0.0.

--flows=*MIN*

Add **--threshold=records**=*MIN* to the options. Deprecated since SiLK 3.17.0.

--flows=*MIN-MAX*

Add **--threshold=records**=*MIN-MAX* to the options. Deprecated since SiLK 3.17.0.

--sip-distinct

Append **Distinct:sIP** to the argument of the **--values** switch unless it is already present. Deprecated since SiLK 2.0.0.

--sip-distinct=*MIN*

Add **--threshold=distinct:sip**=*MIN* to the options. Deprecated since SiLK 3.17.0.

--sip-distinct=*MIN-MAX*

Add **--threshold=distinct:sip**=*MIN-MAX* to the options. Deprecated since SiLK 3.17.0.

--dip-distinct

Append **Distinct:dIP** to the argument of the **--values** switch unless it is already present. Deprecated since SiLK 2.0.0.

--dip-distinct=*MIN*

Add **--threshold=distinct:dip**=*MIN* to the options. Deprecated since SiLK 3.17.0.

--dip-distinct=*MIN-MAX*

Add **--threshold=distinct:dip**=*MIN-MAX* to the options. Deprecated since SiLK 3.17.0.

--stime

Append **sTime-Earliest** to the argument of the **--values** switch unless it is already present. Deprecated since SiLK 2.0.0.

--etime

Append **eTime-Latest** to the argument of the **--values** switch unless it is already present. Deprecated since SiLK 2.0.0.

EXAMPLES

In these examples, the dollar sign (`$`) represents the shell prompt and a backslash (\) is used to continue a line for better readability. Many examples assume previous **rwfilter(1)** commands have written data files named *data.rw* and *data-v6.rw*.

The **--fields** switch is required to specify which field(s) comprise the key. By default, **rwuniq** counts the number of records for each key. This example uses the source port as the key.

```
$ rwuniq --fields=sport data.rw | head
sPort|   Records|
   53|     62216|
   22|     27994|
   67|      7807|
29897|        78|
28816|        24|
   80|     27044|
28925|        22|
    0|      7801|
29246|        63|
```

Notice how the keys are printed in an arbitrary order. Use the **--sort-output** switch to arrange the keys from lowest to highest.

```
$ rwuniq --fields=sport --sort-output data.rw | head
sPort|   Records|
    0|      7801|
   22|     27994|
   25|     15568|
   53|     62216|
   67|      7807|
   80|     27044|
  123|      7741|
  443|      7917|
 8080|      3946|
```

To sort the output by a volume field (such as the number of records), use **rwstats(1)**.

```
$ rwstats --fields=sport --count=10 data.rw
INPUT: 250928 Records for 4739 Bins and 250928 Total Records
OUTPUT: Top 10 Bins by Records
sPort|   Records|  %Records|   cumul_%|
   53|     62216| 24.794363| 24.794363|
   22|     27994| 11.156188| 35.950552|
   80|     27044| 10.777594| 46.728145|
   25|     15568|  6.204170| 52.932315|
```

```
  443|      7917|  3.155088| 56.087404|
   67|      7807|  3.111251| 59.198655|
    0|      7801|  3.108860| 62.307515|
  123|      7741|  3.084949| 65.392463|
 8080|      3946|  1.572563| 66.965026|
29921|       117|  0.046627| 67.011653|
```

Alternatively, process the textual output of **rwuniq** with the UNIX **sort(1)** utility.

```
$ rwuniq --fields=sport data.rw  \
  | sort -r -t '|' -k 2 | head
sPort|   Records|
   53|     62216|
   22|     27994|
   80|     27044|
   25|     15568|
  443|      7917|
   67|      7807|
    0|      7801|
  123|      7741|
 8080|      3946|
```

Use the **--values** field to change the volume that **rwuniq** computes for each key. This example prints the byte-, packet-, and record-counts for each protocol, sorting the results by protocol.

```
$ rwuniq --fields=proto --values=bytes,packets,records --sort data.rw
pro|               Bytes|        Packets|   Records|
  1|             5344836|          73473|      7801|
  6|         59945492930|       72127917|    165363|
 17|            17553593|          77764|     77764|
```

The **--threshold** switch limits the output to rows where a value field meets a minimum value or falls within a specific range. For example, print the number of records and packets seen for each source port for bins having at least 1000 records.

```
$ rwuniq --fields=sport --values=records,packets \
       --threshold=records=1000 data.rw
sPort|   Records|        Packets|
   53|     62216|          62216|
   22|     27994|       23434615|
   67|      7807|           7807|
   80|     27044|        8271125|
    0|      7801|          73473|
  123|      7741|           7741|
   25|     15568|         427777|
  443|      7917|        2421124|
 8080|      3946|        1202528|
```

Multiple thresholds may be specified.

```
$ rwuniq --fields=sport --values=records,packets                 \
       --threshold=records=1000-5000 --threshold=packets=1000000 \
       data.rw
sPort|   Records|        Packets|
 8080|      3946|        1202528|
```

The **--bin-time** switch adjusts the times used by the `sTime` and `eTime` key fields. An argument of 86400 moves the starting and ending time to day boundaries.

```
$ rwuniq --bin-time=86400 --fields=stime,etime data.rw
              sTime|              eTime|   Records|
2009/02/12T00:00:00|2009/02/12T00:00:00|     82969|
2009/02/12T00:00:00|2009/02/13T00:00:00|       360|
2009/02/13T00:00:00|2009/02/13T00:00:00|     83594|
2009/02/13T00:00:00|2009/02/14T00:00:00|       332|
2009/02/14T00:00:00|2009/02/14T00:00:00|     83673|
```

The **--bin-time** switch does not adjust the `duration` value unless both `sTime` and `eTime` are given.

```
$ rwuniq --bin-time=86400 --fields=stime,dur --sort data.rw | head -6
              sTime|durat|   Records|
2009/02/12T00:00:00|    0|     29523|
2009/02/12T00:00:00|    1|      4312|
2009/02/12T00:00:00|    2|      4376|
2009/02/12T00:00:00|    3|      3986|
2009/02/12T00:00:00|    4|       923|
```

```
$ rwuniq --bin-time=86400 --fields=stime,dur,etime data.rw
              sTime|durat|              eTime|   Records|
2009/02/12T00:00:00|    0|2009/02/12T00:00:00|     82969|
2009/02/12T00:00:00|86400|2009/02/13T00:00:00|       360|
2009/02/13T00:00:00|    0|2009/02/13T00:00:00|     83594|
2009/02/13T00:00:00|86400|2009/02/14T00:00:00|       332|
2009/02/14T00:00:00|    0|2009/02/14T00:00:00|     83673|
```

As of SiLK 3.17.0, the **--bin-time** switch accepts a floating point value. When the fractional part is non-zero, **rwuniq** uses millisecond precision for the times and the duration.

```
$ rwuniq --bin-time=0.001 --fields=duration data.rw | head -6
 duration|   Records|
    0.000|     85565|
 1791.045|         4|
    2.120|        19|
   22.263|         5|
   19.902|         3|
```

The **--bin-time** does not adjust the `sTime-Earliest` and `eTime-Latest` aggregate value fields, but it does determine whether those fields maintain millisecond precision.

```
$ rwuniq --bin-time=86400 --fields=stime --value=etime data.rw
              sTime|       eTime-Latest|
2009/02/12T00:00:00|2009/02/12T00:29:59|
2009/02/13T00:00:00|2009/02/13T00:29:58|
2009/02/14T00:00:00|2009/02/14T00:29:59|


$ rwuniq --bin-time=0.001 --fields=proto --value=stime,etime data.rw
pro|         sTime-Earliest|           eTime-Latest|
 17|2009/02/12T00:00:02.745|1970/01/15T06:57:35.997|
  6|2009/02/12T00:00:03.004|1970/01/15T06:57:35.998|
  1|2009/02/12T00:00:20.601|1970/01/15T06:57:35.992|
```

With an input of both IPv4 and IPv6 records, **rwuniq** maps the IPv4 records into the ::ffff:0:0/96 netblock. The data is normally mapped back to IPv4 on output. Given this input:

```
$ rwcut --fields=sip,packets /tmp/v4v6.rw
                                     sIP|   packets|
                                     ::1|        45|
                                192.0.2.22|        87|
                    ::ffff:203.0.113.113|      2662|
                  2001:db8:54:32:ab:cd::|       345|
```

The **rwuniq** tool produces:

```
$ rwuniq --fields=sip --values=packets /tmp/v4v6.rw
                                     sIP|        Packets|
                                     ::1|             45|
                                192.0.2.22|             87|
                               203.0.113.113|           2662|
                  2001:db8:54:32:ab:cd::|            345|
```

Set the **--ip-format** to map-v4 to leave the values as IPv4-mapped IPv6. (Using an **--ipv6-policy** of `force-ipv6` has the same effect.)

```
$ rwuniq --fields=sip --values=packets --ip-format=map-v4 /tmp/v4v6.rw
                                     sIP|        Packets|
                                     ::1|             45|
                         ::ffff:192.0.2.22|             87|
                    ::ffff:203.0.113.113|           2662|
                  2001:db8:54:32:ab:cd::|            345|
```

Print the source addresses that sent more than 10,000,000 bytes, and for each address print the number of unique destination hosts it contacted:

```
$ rwuniq --fields=sip --values=bytes,distinct:dip \
       --threshold=bytes=10000000 data-v6.rw
                      sIP|               Bytes|dIP-Distin|
     2001:db8:a:fd::90:bd|            14529210|         2|
```

Print the number of bytes that host shared with each destination (first use **rwfilter** to limit the input to that host):

```
$ rwfilter --saddr=2001:db8:a:fd::90:bd --pass=- data-v6.rw        \
  | rwuniq --fields=dip --values=bytes
                     dIP|               Bytes|
    2001:db8:c0:a8::fa:5d|             7097847|
     2001:db8:c0:a8::dd:6|             7431363|
```

Print the packet and byte counts for each IPv4 source-destination pair, where the prefix length is 16 (use **rwnetmask(1)** on the input to **rwuniq**):

```
$ rwnetmask --4sip-prefix=16 --4dip-prefix=16 data.rw      \
  | rwuniq --fields=sip,dip --values=packet,byte | head
           sIP|            dIP|  Packets|        Bytes|
    10.139.0.0|    192.168.0.0|    33490|     22950353|
     10.40.0.0|    192.168.0.0|      258|        18544|
    10.204.0.0|    192.168.0.0|   353233|    288736424|
    10.106.0.0|    192.168.0.0|    13051|      3843693|
     10.71.0.0|    192.168.0.0|     4355|      1391194|
     10.98.0.0|    192.168.0.0|     7312|      7328359|
    10.114.0.0|    192.168.0.0|     2538|      4137927|
    10.168.0.0|    192.168.0.0|    92094|     86883062|
    10.176.0.0|    192.168.0.0|   122101|    116555051|
```

Given a file of scan traffic, print the source of TCP traffic with no more than 3 packets and which also appears at least 4 times. First use **rwfilter** to limit the traffic to TCP and find the flow records where the packet count in that flow record is no more than 3.

```
$ rwfilter --proto=6 --packets=1-3 --pass=- scandata.rw          \
  | rwuniq --field=sip --values=flow,packets --threshold=flows=4 \
  | head -5
            sIP|   Records|        Packets|
  10.249.216.38|       256|            256|
   10.155.55.93|       256|            256|
  10.61.255.154|       256|            256|
   10.60.122.82|       256|            256|
```

The **silkpython(3)** manual page provides examples that use PySiLK to create arbitrary fields to use as part of the key for **rwuniq**.

When using **rwuniq** on input that contains both incoming and outgoing flow records, consider using the **int-ext-fields(3)** plug-in which defines four additional fields representing the external IP address, the external port, the internal IP address, and the internal port. The plug-in requires the user to specify which class/type pairs are incoming and which are outgoing. See its manual page for additional information. As an example, here we run **rwuniq** on a file containing incoming and outgoing web traffic.

```
$ rwuniq --fields=sip,sport,dip,dport --values=bytes \
       --sort-output data.rw | head -7
            sIP|sPort|            dIP|dPort|               Bytes|
    10.4.52.235|29631|192.168.233.171|   80|               18260|
```

```
   10.5.231.251|   80|192.168.226.129|28770|              536169|
    10.9.77.117|29906| 192.168.184.65|   80|               55386|
    10.11.88.88|   80|192.168.251.222|28902|              433198|
  10.14.110.214|29989| 192.168.249.96|   80|               25903|
   10.15.224.27|  443| 192.168.231.49|29779|              163759|
```

Here the **int-ext-fields** plug-in is used:

```
$ export INCOMING_FLOWTYPES=all/in,all/inweb
$ export OUTGOING_FLOWTYPES=all/out,all/outweb
$ rwuniq --plugin=int-ext-fields.so \
       --fields=ext-ip,ext-port,int-ip,int-port --value=bytes \
       --sort-output data.rw | head -7
         ext-ip|ext-p|         int-ip|int-p|               Bytes|
    10.4.52.235|29631|192.168.233.171|   80|              726111|
   10.5.231.251|   80|192.168.226.129|28770|              561654|
    10.9.77.117|29906| 192.168.184.65|   80|             1811738|
    10.11.88.88|   80|192.168.251.222|28902|              444277|
  10.14.110.214|29989| 192.168.249.96|   80|              393068|
   10.15.224.27|  443| 192.168.231.49|29779|              167696|
```

ENVIRONMENT

SILK_IPV6_POLICY

This environment variable is used as the value for **--ipv6-policy** when that switch is not provided.

SILK_IP_FORMAT

This environment variable is used as the value for **--ip-format** when that switch is not provided. *Since SiLK 3.11.0.*

SILK_TIMESTAMP_FORMAT

This environment variable is used as the value for **--timestamp-format** when that switch is not provided. *Since SiLK 3.11.0.*

SILK_PAGER

When set to a non-empty string, **rwuniq** automatically invokes this program to display its output a screen at a time. If set to an empty string, **rwuniq** does not automatically page its output.

PAGER

When set and SILK_PAGER is not set, **rwuniq** automatically invokes this program to display its output a screen at a time.

SILK_TMPDIR

When set and **--temp-directory** is not specified, **rwuniq** writes the temporary files it creates to this directory. SILK_TMPDIR overrides the value of TMPDIR.

TMPDIR

When set and SILK_TMPDIR is not set, **rwuniq** writes the temporary files it creates to this directory.

PYTHONPATH

This environment variable is used by Python to locate modules. When **--python-file** is specified, **rwuniq** must load the Python files that comprise the PySiLK package, such as *silk/__init__.py*. If this *silk/* directory is located outside Python's normal search path (for example, in the SiLK installation tree), it may be necessary to set or modify the PYTHONPATH environment variable to include the parent directory of *silk/* so that Python can find the PySiLK module.

SILK_PYTHON_TRACEBACK

When set, Python plug-ins print traceback information on Python errors to the standard error.

SILK_COUNTRY_CODES

This environment variable allows the user to specify the country code mapping file that **rwuniq** uses when computing the scc and dcc fields. The value may be a complete path or a file relative to the SILK_PATH. See the FILES section for standard locations of this file.

SILK_ADDRESS_TYPES

This environment variable allows the user to specify the address type mapping file that **rwuniq** uses when computing the sType and dType fields. The value may be a complete path or a file relative to the SILK_PATH. See the FILES section for standard locations of this file.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, **rwuniq** may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files and plug-ins, **rwuniq** may use this environment variable. See the FILES section for details.

TZ

When the argument to the **--timestamp-format** switch includes `local` or when a SiLK installation is built to use the local timezone, the value of the TZ environment variable determines the timezone in which **rwuniq** displays timestamps. (If both of those are false, the TZ environment variable is ignored.) If the TZ environment variable is not set, the machine's default timezone is used. Setting TZ to the empty string or 0 causes timestamps to be displayed in UTC. For system information on the TZ variable, see **tzset(3)** or **environ(7)**. (To determine if SiLK was built with support for the local timezone, check the `Timezone support` value in the output of **rwuniq --version**.)

SILK_PLUGIN_DEBUG

When set to 1, **rwuniq** prints status messages to the standard error as it attempts to find and open each of its plug-ins. In addition, when an attempt to register a field fails, **rwuniq** prints a message specifying the additional function(s) that must be defined to register the field in **rwuniq**. Be aware that the output can be rather verbose.

SILK_TEMPFILE_DEBUG

When set to 1, **rwuniq** prints debugging messages to the standard error as it creates, re-opens, and removes temporary files.

SILK_UNIQUE_DEBUG

When set to 1, the binning engine used by **rwuniq** prints debugging messages to the standard error.

FILES

${SILK_ADDRESS_TYPES}

${SILK_PATH}/share/silk/address_types.pmap

${SILK_PATH}/share/address_types.pmap

/usr/local/share/silk/address_types.pmap

/usr/local/share/address_types.pmap

Possible locations for the address types mapping file required by the sType and dType fields.

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

${SILK_COUNTRY_CODES}

${SILK_PATH}/share/silk/country_codes.pmap

${SILK_PATH}/share/country_codes.pmap

/usr/local/share/silk/country_codes.pmap

/usr/local/share/country_codes.pmap

Possible locations for the country code mapping file required by the scc and dcc fields.

${SILK_PATH}/lib64/silk/

${SILK_PATH}/lib64/

${SILK_PATH}/lib/silk/

${SILK_PATH}/lib/

/usr/local/lib64/silk/

/usr/local/lib64/

/usr/local/lib/silk/

/usr/local/lib/

Directories that **rwuniq** checks when attempting to load a plug-in.

${SILK_TMPDIR}/

${TMPDIR}/

/tmp/

Directory in which to create temporary files.

NOTES

If multiple thresholds are given (e.g., `--threshold=bytes=80 --threshold=flows=2`), the values must meet all thresholds before the record is printed. For example, if a given key saw a single 100-byte flow, the entry would not printed given the switches above.

rwuniq functionally replaces the combination of

```
rwcut | sort | uniq -c
```

To get a list of unique IP addresses in a data set without the counting or threshold abilities of **rwuniq**, consider using the IPset tools **rwset(1)** and **rwsetcat(1)** for improved performance:

```
rwset --sip-set=stdout | rwsetcat --print-ips
```

For situations where the key and value are each a single field, the Bag tools (**rwbag(1)**, **rwbagcat(1)**) often provide better performance, especially when the key length is one or two bytes:

```
rwbag --bag-file=sport,bytes,stdout | rwbagcat
```

To create a binary file that contains **rwuniq**-like output, use **rwaggbag(1)** or **rwaggbagbuild(1)**. The content of these files may be printed with **rwaggbagcat(1)**.

rwgroup(1) works similarly to **rwuniq**, except the data remains in the form of SiLK Flow records, and the next-hop-IP field is modified to denote the records that form a bin.

rwstats(1) can do the same binning as **rwuniq**, and then sort the data by an aggregate field.

When the **--bin-time** switch is given and the three time fields (starting-time (`sTime`), ending-time (`eTime`), and duration (`duration`)) are present in the key, the duration field's value will be modified to be the difference between the ending and starting times.

When the three time-related key fields (`sTime`,`duration`,`eTime`) are all in use, **rwuniq** will ignore the final time field when binning the records, but the field will appear in the output. Due to truncation of the milliseconds values, **rwuniq** will print a different number of rows depending on the order in which those three values appear in the **--fields** switch.

rwuniq supports counting distinct source and/or destination IPs. To see the number of distinct sources for each 10 minute bin, run:

```
rwuniq --fields=stime --values=distinct:sip --bin-time=600 --sort-output
```

When computing distinct counts over a field, the field may not be part of the key; that is, you cannot have `--fields=sip --values=sip-distinct`.

Using the **--presorted-input** switch sometimes introduces more issues than it solves, and **--presorted-input** is less necessary now that **rwuniq** can use temporary files while processing input.

When computing distinct IP counts, **rwuniq** will typically run faster if you do *not* use the **--presorted-input** switch, even if the data was previously sorted.

When using the **--presorted-input** switch, it is highly recommended that you use no more than one time-related key field (`sTime`, `duration`, `eTime`) in the **--fields** switch and that the time-related key appear last in **--fields**. The issue is caused by **rwsort** considering the millisecond values on the times when sorting, while **rwuniq** truncates the millisecond value. The result may be unsorted output and multiple rows in the output that have the same values for the key fields:

```
$ rwsort --fields=stime,duration data.rw       \
  | rwuniq --fields=stime,dur --presorted
             sTime|durat|   Records|
...
2009/02/12T00:00:57|    0|         2|
2009/02/12T00:00:57|   29|         2|
2009/02/12T00:00:57|    0|         2|
2009/02/12T00:00:57|   13|         2|
...
```

rwuniq's strength is its ability to build arbitrary keys and aggregate fields. For a key of a single IP address, see **rwaddrcount(1)** and **rwbag(1)**; for a key made up of a single CIDR block (/8, /16, /24 only), a single port, or a single protocol, use **rwtotal(1)** or **rwbag(1)**.

As of SiLK 3.17.0, fields that are specified with the legacy thresholding switches (e.g., **--bytes**) and not with **--values** are printed in the order in which those switches appear. Previously, the order was always bytes, packets, flows, stime, etime, sip-distinct, dip-distinct.

SEE ALSO

rwfilter(1), **rwbag(1)**, **rwbagcat(1)**, **rwaggbag(1)**, **rwaggbagbuild(1)**, **rwaggbagcat(1)**, **rwcut(1)**, **rwset(1)**, **rwsetcat(1)**, **rwaddrcount(1)**, **rwgroup(1)**, **rwstats(1)**, **rwnetmask(1)**, **rwsort(1)**, **rwtotal(1)**, **rwcount(1)**, **rwpmapbuild(1)**, **addrtype(3)**, **ccfilter(3)**, **int-ext-fields(3)**, **pmapfilter(3)**, **pysilk(3)**, **silkpython(3)**, **silk-plugin(3)**, **sensor.conf(5)**, **rwflowpack(8)**, **silk(7)**, **yaf(1)**, **dlopen(3)**, **tzset(3)**, **environ(7)**

silk_config

Print SiLK compiling and linking information

SYNOPSIS

```
silk_config [--silk-version] [--compiler] [--cflags] [--include]
       [--libs] [--libsilk-libs] [--libsilk-thrd-libs]
       [--libflowsource-libs] [--data-rootdir] [--python-site-dir]

silk_config --help

silk_config --version
```

DESCRIPTION

silk_config prints configuration information used to compile and link other files and programs against the SiLK header files and libraries. **silk_config** will print the output value(s) selected by the user, or all configuration information if no switches are provided.

This command has nothing to do with the SiLK Configuration file. See the **silk.conf(5)** manual page for information on that file.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option.

--silk-version

Print the version of SiLK as a simple string. This output from this switch is only the version number; the output does not include the additional configuration information that the **--version** switch normally prints.

--compiler

Print the compiler used to build SiLK.

--cflags

Print the include paths (that is, the **-I** switches) and any additional compiler flags to use when compiling a file against the SiLK header files. To only print the include paths, use **--include**.

--include

Print the include paths to use when compiling a file against the SiLK header files. See also **--cflags**.

--libs

This switch is an alias for **--libsilk-libs**.

--libsilk-libs

Print the linker flags (that is, the **-L** and **-l** switches) to use when linking a program against *libsilk.so*.

--libsilk-thrd-libs

Print the linker flags to use when linking a program against *libsilk-thrd.so*. Few external programs will need to use this library.

--libflowsource-libs

Print the linker flags to use when linking a program against **libflowsource.so**. It is highly unlikely that an external program will need to use this library.

--data-rootdir

Print the compiled-in value of the default location of the SiLK data repository, ignoring any environment variable settings.

--python-site-dir

Print the name of the directory containing the *silk* subdirectory where the PySiLK module files were installed. The user may need to set the PYTHONPATH environment variable to this location to be able to use PySiLK. The value will be empty if PySiLK support is not available in this build of SiLK.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

SEE ALSO

silk.conf(5), **silk(7)**

3

SiLK Libraries and Plug-Ins

The behavior of several SiLK tools can be augmented by built-in libraries or plug-ins loaded at run time; this section describes those libraries and plug-ins.

addrtype

Labeling IPv4 addresses as internal or external

SYNOPSIS

```
rwfilter [--stype=ID] [--dtype=ID] ...

rwcut --fields=sType,dType ...

rwgroup --id-fields=sType,dType ...

rwsort --fields=sType,dType ...

rwstats --fields=sType,dType ...

rwuniq --fields=sType,dType ...
```

DESCRIPTION

The *address type* mapping file provides a way to map an IPv4 address to an integer denoting the IP as internal, external, or non-routable. With this mapping file, SiLK flow records can be partitioned (**rwfilter(1)**), displayed (**rwcut(1)**), grouped (**rwgroup(1)**), sorted (**rwsort(1)**), and counted (**rwstats(1)** and **rwuniq(1)**) by the characteristic of the address.

The address type is a specialized form of the Prefix Map, **pmapfilter(3)**, where the following labels are assumed to exist and to have the indicated values:

0
: denotes a (*non-routable*) IP address

1
: denotes an IP address *internal* to the monitored network

2
: denotes an IP address *external* to the monitored network

The SiLK tools look for the address type mapping file in a standard location as detailed in the FILES section below. To provide an alternate location, specify that location in the SILK_ADDRESS_TYPES environment variable.

Creating the prefix map file that maps IPs to one of these labels is described in the MAPPING FILE section below.

OPTIONS

The address type utility provides the following options to the indicated applications.

rwfilter Switches

--stype=*ID*

When *ID* is 0, pass the record if its source address is non-routable. When *ID* is 1, pass the record if its source address is internal. When *ID* is 2, pass the record if its source address is external (i.e., routable and not internal). When *ID* is 3, pass the record if its source address is not internal (non-routable or external).

--dtype=*ID*

As **--stype** for the destination IP address.

rwcut, rwgroup, rwsort, rwstats, and rwuniq Switches

--fields=*FIELDS*

FIELDS refers to a list of fields to use for the operation. The address type utility makes two additional fields, `sType` (alias `16`) and `dType` (`17`) available for display, grouping, sorting, and counting using the **rwcut(1)**, **rwgroup(1)**, **rwsort(1)**, **rwstats(1)**, and **rwuniq(1)** tools:

sType,16

For the source IP address, prints 0 if the address is non-routable, 1 if it is internal, or 2 if it is routable and external.

dType,17

as **sType**, except for the destination address

MAPPING FILE

To denote an address as `non-routable`, `internal`, or `external` at your site, you will need to create the *address_types.pmap* file and either install it in the appropriate location (see the FILES section below) or set the SILK_ADDRESS_TYPES environment variable to the file's location.

The **rwpmapbuild(1)** tool creates a *prefix map* file from a text file. A template for the text file is available in $SILK_PATH/share/silk/addrtype-templ.txt. The text file used to create *address_types.pmap* must include the following section to ensure that IPs are mapped to the integer values that the *addrtype.so* expects:

```
#    Numerical mappings of labels

label 0             non-routable
label 1             internal
label 2             external

#    Default to "external" for all un-defined ranges.

default             external
```

The remainder of the file can list CIDR blocks and a label for each block:

```
# RFC1918 space
10.0.0.0/8          non-routable
172.16.0.0/12       non-routable
192.168.0.0/16      non-routable
```

```
# My IP space (CMU)
128.2.0.0/16        internal
```

Once the text file is saved to disk, use **rwpmapbuild** to create *address_types.pmap*:

```
rwpmapbuild --input addresses.txt --output address_types.pmap
```

ENVIRONMENT

SILK_ADDRESS_TYPES

This environment variable allows the user to specify the address type mapping file to use. The value may be a complete path or a file relative to SILK_PATH. If the variable is not specified, the code looks for a file named *address_types.pmap* as specified in the FILES section below.

SILK_PATH

This environment variable gives the root of the install tree. The SiLK applications check the directories *$SILK_PATH/share/silk* and *$SILK_PATH/share* for the address type mapping file, *address_types.pmap*.

FILES

The tools will look for the data file that maps IPs to labels in the following locations. ($SILK_ADDRESS_TYPES is the value of the SILK_ADDRESS_TYPES environment variable, if it is set. $SILK_PATH is value of the SILK_PATH environment variable, if it is set. The use of */usr/local/* assumes the application is installed in the */usr/local/bin/* directory.)

```
$SILK_ADDRESS_TYPES
$SILK_PATH/share/silk/address_types.pmap
$SILK_PATH/share/address_types.pmap
/usr/local/share/silk/address_types.pmap
/usr/local/share/address_types.pmap
```

SEE ALSO

rwcut(1), **rwfilter(1)**, **rwgroup(1)**, **rwpmapbuild(1)**, **rwpmapcat(1)**, **rwsort(1)**, **rwstats(1)**, **rwuniq(1)**, **pmapfilter(3)**, **silk(7)**

app-mismatch

SiLK plug-in to find services on unusual ports

SYNOPSIS

```
rwfilter --plugin=app-mismatch.so ...
```

DESCRIPTION

The **app-mismatch** plug-in adds a partitioning rule to **rwfilter(1)** that helps to find services running on unusual port numbers.

Specifically, when the **app-mismatch** plug-in is loaded into **rwfilter(1)**, **rwfilter** adds a partitioning rule that passes a record when the record's application field (the **applabel(1)** value determined by **yaf(1)**) is set and the value does not match the value of either the source port or destination port.

The plug-in causes **rwfilter** to write each record that meets any of these criteria to the location specified by the **--fail-destination** switch:

- the `protocol` field has a value other than 6 or 17 (TCP or UDP)
- the `application` field has the value 0, indicating that the application labeling feature was disabled or that it was unable to determine the type of application
- the `application` field value is equal to either the `sPort` or the `dPort` field, indicating the type of traffic appears to be consistent with what would be expected

The remaining records are either TCP or UDP records where the application field is set and its value is different than that in the source and destination port. These records are written to the location specified by the **--pass-destination** switch.

OPTIONS

The **app-mismatch** plug-in does not add any additional switches to **rwfilter** nor modify any field.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

The *app-mismatch.so* plug-in must be explicitly loaded into **rwfilter(1)** using the **--plugin** switch. The plug-in becomes active once it is loaded and no additional switches are required.

The following searches the SiLK Flow file *data.rw* for services that appear to be running on unusual or non-typical ports. To get a quick summary of the data, the output from **rwfilter** is piped into **rwuniq(1)**:

```
$ rwfilter --plugin=app-mismatch.so --print-stat --pass=- data.rw \
  | rwuniq --fields=application,sPort,dPort | head
```

```
Files     1.  Read      24494.  Pass        890. Fail       23604.
appli|sPort|dPort|   Records|
   53|62579| 5355|         1|
   53|55188| 5355|         1|
   53|57807| 5355|         1|
   53|54898| 5355|         1|
   80| 1171|  591|         1|
   53| 5355|50478|         1|
   53|64981| 5355|         1|
  139|52845|  445|         1|
   53|52536| 5355|         1|
```

As seen in the output of the **--print-stat** switch from **rwfilter**, the plug-in failed 23,604 records. Some of those records have protocols other than TCP and UDP, and some records have an application value of zero. Adding additional **rwfilter** invocations provides a way to get count for each:

```
$ rwfilter --protocol=6,17 --print-stat --pass=- data.rw       \
  | rwfilter --application=1- --print-stat --pass=- -          \
  | rwfilter --plugin=app-mismatch.so --print-stat --pass=- -  \
  | rwuniq --fields=application,sPort,dPort --pager= | head
Files     1.  Read      24494.  Pass      24420. Fail          74.
Files     1.  Read      24420.  Pass      14228. Fail       10192.
Files     1.  Read      14228.  Pass        890. Fail       13338.
appli|sPort|dPort|   Records|
   53|62579| 5355|         1|
   53|55188| 5355|         1|
   53|57807| 5355|         1|
   53|54898| 5355|         1|
   80| 1171|  591|         1|
   53| 5355|50478|         1|
   53|64981| 5355|         1|
  139|52845|  445|         1|
   53|52536| 5355|         1|
```

All but 74 records were either TCP or UDP. For the TCP and UDP records, 10,192 had an application label of 0. There were 13,338 records where the application label matched the port number. Change the final **rwfilter** invocation to use **--fail-destination** to see those records:

```
$ rwfilter --protocol=6,17 --print-stat --pass=- data.rw       \
  | rwfilter --application=1- --print-stat --pass=- -          \
  | rwfilter --plugin=app-mismatch.so --print-stat --pass=- -  \
  | rwuniq --fields=application,sPort,dPort --pager= | head
Files     1.  Read      24494.  Pass      24420. Fail          74.
Files     1.  Read      24420.  Pass      14228. Fail       10192.
Files     1.  Read      14228.  Pass        890. Fail       13338.
appli|sPort|dPort|   Records|
  443|  443|53257|         1|
   80|54123|   80|         2|
   80|52322|   80|         1|
   80|54749|   80|         1|
   80|   80|52885|         3|
```

```
80|   80|54204|         1|
53|   53|55964|         1|
80|53497|   80|         1|
80|54122|   80|         2|
```

ENVIRONMENT

SILK_PATH

This environment variable gives the root of the install tree. When searching for plug-ins, a SiLK application may use this environment variable. See the FILES section for details.

SILK_PLUGIN_DEBUG

When set to 1, the SiLK applications print status messages to the standard error as they attempt to find and open the *app-mismatch.so* plug-in. A typical invocation using this variable is:

```
env SILK_PLUGIN_DEBUG=1 rwfilter --plugin=app-mismatch.so --version
```

FILES

${SILK_PATH}/lib64/silk/app-mismatch.so

${SILK_PATH}/lib64/app-mismatch.so

${SILK_PATH}/lib/silk/app-mismatch.so

${SILK_PATH}/lib/app-mismatch.so

/usr/local/lib64/silk/app-mismatch.so

/usr/local/lib64/app-mismatch.so

/usr/local/lib/silk/app-mismatch.so

/usr/local/lib/app-mismatch.so

Possible locations for the plug-in.

SEE ALSO

rwfilter(1), **rwuniq(1)**, **silk(7)**, **yaf(1)**, **applabel(1)**

ccfilter

Mapping IPv4 addresses to country codes

SYNOPSIS

```
rwfilter [--scc=COUNTRY_CODES] [--dcc=COUNTRY_CODES] ...

rwcut --fields=scc,dcc ...

rwgroup --id-fields=scc,dcc ...

rwsort --fields=scc,dcc ...

rwstats --fields=scc,dcc ...

rwuniq --fields=scc,dcc ...

rwpmaplookup --country-codes ...
```

DESCRIPTION

The *country code* mapping file provides a mapping from an IPv4 address to two-letter, lowercase abbreviation of the country where that IP address is located. The mapping file allows the country code value of IP addresses on a SiLK Flow record to be partitioned (**rwfilter(1)**), displayed (**rwcut(1)**), sorted (**rwsort(1)**), grouped (**rwgroup(1)**), and counted (**rwstats(1)** and **rwuniq(1)**).

The **rwpmaplookup(1)** tool, when invoked with the **--country-codes** switch, accepts textual input and prints the country code for the IPs, which provide a way to print country codes for the IPs in SiLK IPsets or bags.

The abbreviations used by the country code utility are the two-letter codes defined in ISO 3166 part 1. For additional information, see https://www.iso.org/iso-3166-country-codes.html and https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2. Some IP addresses map to one of the following special codes:

--

N/A (e.g. private and experimental reserved addresses)

a1

anonymous proxy

a2

satellite provider

o1

other

The SiLK tools look for the country code mapping file in a standard location as detailed in the FILES section below. To provide an alternate location, specify that location in the SILK_COUNTRY_CODES environment variable.

Creating the Prefix Map (pmap) file that maps an IP to its country code requires the GeoIP2 Country or free GeoLite2 database created by MaxMind, available from https://dev.maxmind.com/geoip/, as described in the MAPPING FILE section below.

OPTIONS

Country code support makes available two additional keys to the **--fields** switch in the **rwcut(1)**, **rwgroup(1)**, **rwsort(1)**, **rwstats(1)**, and **rwuniq(1)** tools:

scc,18

Print, sort, and/or count the flow records by the country code designation of the source IP address

dcc,19

As **scc** for the destination address

In **rwfilter(1)**, the following switches are supported:

--scc=*COUNTRY_CODE_LIST*

Pass the record if the country code of its source IP address is in the specified *COUNTRY_CODE_LIST*.

--dcc=*COUNTRY_CODE_LIST*

As **--scc** for the destination IP address.

MAPPING FILE

To map from IP addresses to country codes you will need to create the *country_codes.pmap* data file and install it in the appropriate location (see the FILES section below), or specify the path to the file in the SILK_COUNTRY_CODES environment variable.

The prefix map data file is based on the GeoIP2 Country(R) or free GeoLite2 database created by MaxMind and available from https://dev.maxmind.com/geoip/. We do not distribute the database nor the data file, but we provide the **rwgeoip2ccmap(1)** tool that converts the GeoIP database to the format that *ccfilter.so* expects.

MaxMind distributes multiple versions of their GeoIP Country database; one is a free evaluation copy. In addition, they sell versions with higher accuracy, and they offer various subscription services.

ENVIRONMENT

SILK_COUNTRY_CODES

This environment variable allows the user to specify the country code mapping file that the SiLK tools use. The value may be a complete path or a file relative to SILK_PATH. If the variable is not specified, the code looks for a file named *country_codes.pmap* as specified in the FILES section below.

SILK_PATH

This environment variable gives the root of the install tree. The SiLK applications check the directories *$SILK_PATH/share/silk* and *$SILK_PATH/share* for the country code mapping file, *country_codes.pmap*.

FILES

The tools will look for the data file that maps IPs to country codes in the following locations. ($SILK_COUNTRY_CODES is the value of the SILK_COUNTRY_CODES environment variable, if it is set. $SILK_PATH is value of the SILK_PATH environment variable, if it is set. The use of */usr/local/* assumes the application is installed in the */usr/local/bin/* directory.)

```
$SILK_COUNTRY_CODES
$SILK_PATH/share/silk/country_codes.pmap
$SILK_PATH/share/country_codes.pmap
/usr/local/share/silk/country_codes.pmap
/usr/local/share/country_codes.pmap
```

SEE ALSO

rwcut(1), **rwfilter(1)**, **rwgroup(1)**, **rwsort(1)**, **rwstats(1)**, **rwuniq(1)**, **rwgeoip2ccmap(1)**, **rwpmaplookup(1)**, **silk(7)**

conficker-c

SiLK plug-in to detect traffic matching the Conficker C worm

SYNOPSIS

```
rwfilter --plugin=conficker-c.so [--conficker-seed=SEED]
      [--s-conficker] [--d-conficker] [--a-conficker] ...

rwcut --plugin=conficker-c.so [--conficker-seed=SEED]
      [--fields=...,sconficker,dconficker,...] ...

rwgroup --plugin=conficker-c.so [--conficker-seed=SEED]
      [--fields=...,sconficker,dconficker,...] ...

rwsort --plugin=conficker-c.so [--conficker-seed=SEED]
      [--fields=...,sconficker,dconficker,...] ...

rwstats --plugin=conficker-c.so [--conficker-seed=SEED]
      [--fields=...,sconficker,dconficker,...] ...

rwuniq --plugin=conficker-c.so [--conficker-seed=SEED]
      [--fields=...,sconficker,dconficker,...] ...
```

DESCRIPTION

The **conficker-c** plug-in was written in March 2009 to detect traffic that matches the signature of the .C variant of the Conficker worm.

The .C variant of the Conficker worm (https://www.us-cert.gov/ncas/alerts/TA09-088A) contains a peer-to-peer scanning thread which generates a large amount of UDP high-port to high-port packets. SRI International provides a detailed analysis report on the worm's behavior which describes features of the peer-to-peer network traffic. (http://www.csl.sri.com/users/vinod/papers/Conficker/addendumC/index.html) This report hints at "...a unique mapping from IP address to the two TCP and UDP listen ports in each host."

This type of behavior is also ideally suited for flow analysis, and the **conficker-c** plug-in emulates the same functionality. When loaded into either **rwfilter(1)** or **rwcut(1)** using the **--plugin** switch, the plug-in adds fields for detecting and filtering Conficker.C traffic with a limited number of false positives.

The **conficker-c** plug-in identifies the *targets* of Conficker.C scanning. When a Conficker.C infected machine starts scanning for other peers, it targets a somewhat random port on the destination host according to a function **f()** where

```
dPort = f (dIP, seed)
sPort = f (sIP, seed)
```

and the seed is computed from the function **g()**:

```
seed = g (start_time)
```

The plug-in implements that function and can check whether the computed source or destination port matches the observed value of the port. If the source or destination matches, that indicates that the destination or source, respectively, may be infected.

To compute the `seed` argument to the function, the plug-in computes the number of weeks between 1970-Jan-05 and the flow record's start time. When the flow's start time is within a few minutes of the week boundary, the plug-in computes ports using both possible values for the seed. The plug-in provides the **--conficker-seed** command line switch to allow selection of a different seed.

The plug-in ignores any non-UDP/non-TCP traffic.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

The **conficker-c** plug-in provides the following options to the indicated applications.

rwfilter Switches

The **conficker-c** plug-in adds the following switches to **rwfilter(1)**. You may check for Conficker.C traffic on a particular side of the flow, or for both sides:

--s-conficker

Pass the flow record if the source IP and port match those targeted by Conficker.C (indicating that the destination IP may be infected).

--d-conficker

Pass the flow record if the destination IP and port match those targeted by Conficker.C (indicating that the source IP may be infected).

--a-conficker

Pass the flow record if either the source IP and port or the destination IP and port match those targeted by Conficker.C.

--conficker-seed=*SEED*

Use the value *SEED* to seed Conficker.C checker. Typically the flow's start time is used as the basis for the seed.

rwcut, rwgroup, rwsort, rwstats, and rwuniq Switches

The **conficker-c** plug-in augments the switches of **rwcut(1)**, **rwgroup(1)**, **rwsort(1)**, **rwstats(1)**, and **rwuniq(1)** as follows:

--fields=*FIELDS*

FIELDS refers to a list of fields to use for the operation. The **conficker-c** plug-in adds the following fields:

sconficker

Show whether the source IP and source port combination match the values targeted by Conficker.C, which indicate that the destination IP may be infected. This field contains a `1` when values match and a `0` when they do not.

dconficker

Show whether the destination IP and destination port combination match the values targeted by Conficker.C, which indicate that the source IP may be infected. This field contains a `1` when values match and a `0` when they do not.

--conficker-seed=*SEED*

Use the value *SEED* to seed Conficker.C checker. Typically the flow's start time is used as the basis for the seed.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

This example uses contrived data to test that the plug-in works. Values that are known to match the worm are piped into **rwtuc(1)** to create a SiLK Flow record. That record is piped into **rwfilter**, which matches the record. That result is piped into **rwcut** to display the result:

```
$ echo '17|10.10.10.10|23332|192.168.192.168|16514|'        \
  | rwtuc --fields=protocol,sip,sport,dip,dport              \
  | rwfilter --plugin=conficker.so --conficker-seed=8888     \
        --s-conficker --protocol=17 --print-volume           \
        --pass=stdout stdin                                  \
  | rwcut --plugin=conficker.so --conficker-seed=8888        \
        --fields=sip,sport,sconficker,dip,dport,dconficker   \
        --ipv6-policy=ignore
     | Recs  | Packets | Bytes | Files |
Total|      1|        1|      1|      1|
Pass |      1|        1|      1|       |
Fail |      0|        0|      0|       |
        sIP|sPort|scon|            dIP|dPort|dcon|
10.10.10.10|23332|   1|192.168.192.168|16514|   1|
```

To find infected hosts on your network, you typically want to look at outgoing traffic and find instances where source hosts are targeting conficker *destination* IP and port pairs, so you would use the **--d-conficker** switch on **rwfilter**.

To further refine the query and eliminate most false positives, it is useful to eliminate common service ports (the packets from a scanner have sport=ephemeral, dport=conficker-chosen):

```
$ rwfilter --plugin=conficker-c.so --d-conficker                \
       --sport=1024- --dport=1024-                              \
       --start-date=2009/05/01 --end-date=2009/05/31 --type=out  \
       --pass=stdout                                            \
  | rwuniq --fields=sip --flows=10 --sort-output
```

There may be false positives from VPN traffic. Depending on your network, you might want to filter traffic on UDP 500 or 10000.

ENVIRONMENT

SILK_PATH

This environment variable gives the root of the install tree. When searching for plug-ins, a SiLK application may use this environment variable. See the FILES section for details.

SILK_PLUGIN_DEBUG

When set to 1, the SiLK applications print status messages to the standard error as they attempt to find and open the *conficker-c.so* plug-in. A typical invocation using this variable is:

```
env SILK_PLUGIN_DEBUG=1 rwcut --plugin=conficker-c.so --version
```

FILES

${SILK_PATH}/lib64/silk/conficker-c.so

${SILK_PATH}/lib64/conficker-c.so

${SILK_PATH}/lib/silk/conficker-c.so

${SILK_PATH}/lib/conficker-c.so

/usr/local/lib64/silk/conficker-c.so

/usr/local/lib64/conficker-c.so

/usr/local/lib/silk/conficker-c.so

/usr/local/lib/conficker-c.so

Possible locations for the plug-in.

SEE ALSO

rwfilter(1), **rwcut(1)**, **rwgroup(1)**, **rwsort(1)**, **rwstats(1)**, **rwuniq(1)**, **rwtuc(1)**, **silk(7)**

cutmatch

Display value in next-hop field written by rwmatch

SYNOPSIS

```
rwcut --plugin=cutmatch.so --fields=...,match,...  ...
```

DESCRIPTION

The **cutmatch** plug-in creates a field in **rwcut(1)** that provides a more user-friendly representation of the match parameter value that **rwmatch(1)** writes into a SiLK Flow record's next hop IP field.

The **cutmatch** plug-in defines a `match` field that displays the direction of the flow (`->` represents a query and `<-` a response) and the numeric match ID.

OPTIONS

The **cutmatch** plug-in modifies the following switch of **rwcut(1)**:

--fields=*FIELDS*

FIELDS refers to a list of fields to print. The **cutmatch** plug-in adds the following field:

match

Print the direction of the flow (`->` represents a query and `<-` a response) and the numeric match ID

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

Given two files containing unidirectional flow records, use **rwsort(1)** and **rwmatch(1)** to create the file *matched.rw* where a query and its response have been labeled with a unique value in the next-hop IP field. See the **rwmatch** manual page for more information.

```
$ rwsort --fields=1,4,2,3,5,stime incoming.rw > incoming-query.rw
$ rwsort --fields=2,3,1,4,5,stime outgoing.rw > outgoing-response.rw
$ rwmatch --relate=1,2 --relate=4,3 --relate=2,1 --relate=3,4 \
       --relate=5,5 incoming-query.rw outgoing-response.rw matched.rw
```

To use the plug-in, you must explicitly load it into **rwcut(1)** by specifying the **--plugin** switch. You can then include `match` in the list of **--fields** to print:

```
$ rwcut --plugin=cutmatch.so --num-rec=8  \
       --fields=sIP,sPort,match,dIP,dPort,type matched.rw
            sIP|sPort| <->Match#|            dIP|dPort|   type|
```

```
    10.4.52.235|29631|->       1|192.168.233.171|   80|  inweb|
192.168.233.171|   80|<-       1|    10.4.52.235|29631| outweb|
    10.9.77.117|29906|->       2| 192.168.184.65|   80|  inweb|
 192.168.184.65|   80|<-       2|    10.9.77.117|29906| outweb|
  10.14.110.214|29989|->       3| 192.168.249.96|   80|  inweb|
 192.168.249.96|   80|<-       3|  10.14.110.214|29989| outweb|
    10.18.66.79|29660|->       4| 192.168.254.69|   80|  inweb|
 192.168.254.69|   80|<-       4|    10.18.66.79|29660| outweb|
```

This shows external hosts querying the web server (the Match column contains ->) and the web server's responses (<-).

Using the `sIP` and `dIP` fields may be confusing when the file you are examining contains both incoming and outgoing flow records. To make the output from **rwmatch** more clear, consider using the **int-ext-fields(3)** plug-in as well. That plug-in allows you to display the external IPs in one column and the internal IPs in a another column. See its manual page for additional information.

```
$ export INCOMING_FLOWTYPES=all/in,all/inweb
$ export OUTGOING_FLOWTYPES=all/out,all/outweb
$ rwcut --plugin=cutmatch.so --plugin=int-ext-fields.so --num-rec=8 \
     --fields=ext-ip,ext-port,match,int-ip,int-port,proto matched.rw
        ext-ip|ext-p| <->Match#|         int-ip|int-p|   type|
   10.4.52.235|29631|->       1|192.168.233.171|   80|  inweb|
   10.4.52.235|29631|<-       1|192.168.233.171|   80| outweb|
   10.9.77.117|29906|->       2| 192.168.184.65|   80|  inweb|
   10.9.77.117|29906|<-       2| 192.168.184.65|   80| outweb|
  10.14.110.214|29989|->       3| 192.168.249.96|   80|  inweb|
  10.14.110.214|29989|<-       3| 192.168.249.96|   80| outweb|
    10.18.66.79|29660|->       4| 192.168.254.69|   80|  inweb|
    10.18.66.79|29660|<-       4| 192.168.254.69|   80| outweb|
```

ENVIRONMENT

SILK_PATH

This environment variable gives the root of the install tree. When searching for plug-ins, a SiLK application may use this environment variable. See the FILES section for details.

SILK_PLUGIN_DEBUG

When set to 1, the SiLK applications print status messages to the standard error as they attempt to find and open the *cutmatch.so* plug-in. A typical invocation using this variable is:

```
env SILK_PLUGIN_DEBUG=1 rwcut --plugin=cutmatch.so --version
```

FILES

${SILK_PATH}/lib64/silk/cutmatch.so

${SILK_PATH}/lib64/cutmatch.so

${SILK_PATH}/lib/silk/cutmatch.so

${SILK_PATH}/lib/cutmatch.so

/usr/local/lib64/silk/cutmatch.so

/usr/local/lib64/cutmatch.so

/usr/local/lib/silk/cutmatch.so

/usr/local/lib/cutmatch.so

Possible locations for the plug-in.

SEE ALSO

rwcut(1), **rwmatch(1)**, **rwsort(1)**, **int-ext-fields(3)**, **silk(7)**

flowkey

SiLK plug-in providing YAF flow key filter and field

SYNOPSIS

```
rwfilter --plugin=flowkey.so [--flow-key=VALUE_LIST]

rwcut --plugin=flowkey.so --fields=FIELDS ...

rwgroup --plugin=flowkey.so --fields=FIELDS ...

rwsort --plugin=flowkey.so --fields=FIELDS ...

rwstats --plugin=flowkey.so --fields=FIELDS --values=FIELDS ...

rwuniq --plugin=flowkey.so --fields=FIELDS --values=FIELDS ...
```

DESCRIPTION

The YAF *flow key hash* is a numeric value that the **yaf(1)** IPFIX generator computes for every flow record. The flow key hash is computed from the IP protocol, the source and destination IP addresses, the source and destination ports, and the vlan identifier. The **getFlowKeyHash(1)** tool in YAF distribution reads IPFIX data and computes the flow key hash for each flow record.

The **flowkey** plug-in uses the same formula as YAF to compute the flow key hash for a record. The flow key hash may be printed by **rwcut(1)**, may be used as part of the sorting key in **rwsort(1)**, may be used as a grouping key in **rwgroup(1)**, **rwstats(1)**, and **rwuniq(1)**, and may be used as a partitioning criterion in **rwfilter(1)**.

Note that the flow key hash computed by this plug-in may be different than the value computed by YAF:

- When SiLK processes a bi-directional IPFIX record (a *bi-flow*), it splits the record into two uni-directional records and reverses the source and destination fields when it stores the reverse record. The flow key hash for this reverse record is different than that of the forward record. The **getFlowKeyHash** tool has a **--reverse** switch to duplicate this behavior.
- YAF computes the flow key hash using the vlan identifier, but SiLK ignores the vlan ID unless it is explicitly instructed to use it. When SiLK is told to the use the vlan ID, the vlan ID is stored in the the **in** field of the SiLK Flow record. That field normally holds the SNMP ingress value.

  (Instructing SiLK to use the vlan ID depends on whether one is using **rwipfix2silk(1)**, **rwflowpack(8)**, or **flowcap(8)**. For **rwipfix2silk**, run the tool with the **--interface-values=vlan** switch. For **rwflowpack** and **flowcap**, edit the **sensor.conf(5)** file and specify `interface-values vlan` in the probe block where the flow is collected.)
- Even when SiLK has been told to store the vlan identifier in the field normally used for the ingress interface, **rwflowpack** typically does not store that field in the files it creates in the data repository. When reading these files, the **in** field is set to 0. To tell **rwflowpack** to store the field, run it with the command line switch **--pack-interfaces**. To tell **getFlowKeyHash** to ignore the value, specify the **--snmp** switch.

The **flowkey** plug-in must be explicitly loaded into an application via the **--plugin** switch.

OPTIONS

The **flowkey** plug-in provides the following options to the indicated applications.

rwfilter Switches

When the **flowkey** plug-in has been loaded, the following switch is added to **rwfilter**. To pass the filter, the record must pass the test implied by the switch.

--flowkey=*VALUE_LIST*

Check whether the flow key hash of the flow record matches one of the values in *VALUE_LIST*, where *VALUE_LIST* is a comma-separated list of values expressed as either decimal or hexadecimal numbers. Hexadecimal numbers must be preceded by `0x`.

rwcut, rwgroup, rwsort, rwstats, and rwuniq Switch

--fields=*FIELDS*

FIELDS refers to a list of fields to use for the operation. The **flowkey** plug-in adds the following field for display, sorting, and grouping using the **rwcut(1)**, **rwgroup(1)**, **rwsort(1)**, **rwstats(1)**, and **rwuniq(1)** tools:

flowkey

Print, sort by, or group by the flow key hash.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

The file **vlan.pcap** is a packet capture file created by **tcpdump(1)**. The packets in the file include vlan identifiers.

In the following command, **yaf(1)** creates IPFIX flow records from the PCAP file, **rwipfix2silk(1)**) converts the IPFIX records to SiLK Flow records, and **rwcut(1)** prints the SiLK records as text. Note the use of the **--interface-values=vlan** switch on **rwipfix2silk**, and see how the **--plugin** switch is used on **rwcut**. The `flowkey` field contains the flow key hash.

```
$ yaf < vlan.pcap                                              \
  | rwipfix2silk --interface-values=vlan                       \
  | rwcut --plugin=flowkey.so --fields=1-5,in,flowkey,stime    \
       --ipv6=ignore --timestamp=epoch --num-rec=9
          sIP|          dIP|sPort|dPort|pro|   in|   flowkey|         sTime|
 10.128.87.50|     10.0.0.4|32942|   80|  6|    2|2148415270|1252941224.465|
     10.0.0.4| 10.128.87.50|   80|32942|  6|    2|  15775704|1252941224.465|
 10.128.87.50|     10.0.0.4|32942|   80|  6|    2|2148415270|1252941224.505|
 10.128.34.93|     10.0.0.3|41443|46612|  6|    2|2705585162|1252941224.505|
     10.0.0.3| 10.128.34.93|46612|41443|  6|    2|3065308157|1252941224.505|
 10.128.34.93|     10.0.0.3|41442|   21|  6|    2|2705474059|1252941224.465|
     10.0.0.3| 10.128.34.93|   21|41442|  6|    2|  11920380|1252941224.465|
```

```
 10.128.44.78|     10.0.0.4|48081|   80|  6|    2|3144764506|1252941276.278|
     10.0.0.4| 10.128.44.78|   80|48081|  6|    2|  15792091|1252941276.279|
```

Here is the output from **getFlowKeyHash(1)** when it is run with no arguments. The `hash` column is the flow key hash and the `ms` column is the flow's time stamp.

```
$ yaf < vlan.pcap              \
  | getFlowKeyHash             \
  | head -10
          sIP|          dIP|sPort|dPort|pro| vlan|      hash|            ms
 10.128.87.50|     10.0.0.4|32942|   80|  6|    2|2148415270| 1252941224465
 10.128.87.50|     10.0.0.4|32942|   80|  6|    2|2148415270| 1252941224505
 10.128.34.93|     10.0.0.3|41443|46612|  6|    2|2705585162| 1252941224505
 10.128.34.93|     10.0.0.3|41442|   21|  6|    2|2705474059| 1252941224465
 10.128.44.78|     10.0.0.4|48081|   80|  6|    2|3144764506| 1252941276278
 10.128.44.78|     10.0.0.4|48081|   80|  6|    2|3144764506| 1252941276279
 10.128.30.43|     10.0.0.4|20803|   80|  6|    2|1373863487| 1252941276278
 10.128.30.43|     10.0.0.4|20803|   80|  6|    2|1373863487| 1252941276280
 10.128.67.47|     10.0.0.4|10912|   80|  6|    2| 704652091| 1252941276278
```

The **rwcut** output has two records for each bi-flow record in the **getFlowKeyHash** output. The hash values match for every-other record.

Adding the **--reverse** switch to **getFlowKeyHash** produces the following:

```
$ yaf < vlan.pcap              \
  | getFlowKeyHash --reverse   \
  | head -10
          sIP|          dIP|sPort|dPort|pro| vlan|      hash|            ms
 10.128.87.50|     10.0.0.4|32942|   80|  6|    2|  15775704| 1252941224465
 10.128.87.50|     10.0.0.4|32942|   80|  6|    2|  15775704| 1252941224505
 10.128.34.93|     10.0.0.3|41443|46612|  6|    2|3065308157| 1252941224505
 10.128.34.93|     10.0.0.3|41442|   21|  6|    2|  11920380| 1252941224465
 10.128.44.78|     10.0.0.4|48081|   80|  6|    2|  15792091| 1252941276278
 10.128.44.78|     10.0.0.4|48081|   80|  6|    2|  15792091| 1252941276279
 10.128.30.43|     10.0.0.4|20803|   80|  6|    2|  15740716| 1252941276278
 10.128.30.43|     10.0.0.4|20803|   80|  6|    2|  15740716| 1252941276280
 10.128.67.47|     10.0.0.4|10912|   80|  6|    2|  15731147| 1252941276278
```

The values for every-other flow record match nearly match, but things appear to get out of sync.

A different approach is to run **yaf** with the **--uniflow** switch:

```
$ yaf --uniflow < vlan.pcap    \
  | getFlowKeyHash             \
  | head -10
          sIP|          dIP|sPort|dPort|pro| vlan|      hash|            ms
 10.128.87.50|     10.0.0.4|32942|   80|  6|    2|2148415270| 1252941224465
     10.0.0.4| 10.128.87.50|   80|32942|  6|    2|  15775704| 1252941224465
 10.128.87.50|     10.0.0.4|32942|   80|  6|    2|2148415270| 1252941224505
 10.128.34.93|     10.0.0.3|41443|46612|  6|    2|2705585162| 1252941224505
```

```
     10.0.0.3| 10.128.34.93|46612|41443|  6|    2|3065308157| 1252941224505
  10.128.34.93|     10.0.0.3|41442|   21|  6|    2|2705474059| 1252941224465
     10.0.0.3| 10.128.34.93|   21|41442|  6|    2|  11920380| 1252941224465
  10.128.44.78|     10.0.0.4|48081|   80|  6|    2|3144764506| 1252941276278
     10.0.0.4| 10.128.44.78|   80|48081|  6|    2|  15792091| 1252941276279
```

This result exactly matches that from **rwcut**.

When **rwipfix2silk** does not include the **--interface-values=vlan** switch, the result is:

```
$ yaf < vlan.pcap                                              \
  | rwipfix2silk                                               \
  | rwcut --plugin=flowkey.so --fields=1-5,in,flowkey,stime    \
       --ipv6=ignore --timestamp=epoch --num-rec=9
          sIP|          dIP|sPort|dPort|pro|   in|   flowkey|         sTime|
  10.128.87.50|     10.0.0.4|32942|   80|  6|    0|2150512422|1252941224.465|
     10.0.0.4| 10.128.87.50|   80|32942|  6|    0|  13678552|1252941224.465|
  10.128.87.50|     10.0.0.4|32942|   80|  6|    0|2150512422|1252941224.505|
  10.128.34.93|     10.0.0.3|41443|46612|  6|    0|2707682314|1252941224.505|
     10.0.0.3| 10.128.34.93|46612|41443|  6|    0|3063211005|1252941224.505|
```

To get the same result from **getFlowKeyHash**, use the **--snmp** switch:

```
$ yaf --uniflow < vlan.pcap    \
  | getFlowKeyHash --snmp      \
  | head -6
          sIP|          dIP|sPort|dPort|pro| vlan|      hash|            ms
  10.128.87.50|     10.0.0.4|32942|   80|  6|    0|2150512422| 1252941224465
     10.0.0.4| 10.128.87.50|   80|32942|  6|    0|  13678552| 1252941224465
  10.128.87.50|     10.0.0.4|32942|   80|  6|    0|2150512422| 1252941224505
  10.128.34.93|     10.0.0.3|41443|46612|  6|    0|2707682314| 1252941224505
     10.0.0.3| 10.128.34.93|46612|41443|  6|    0|3063211005| 1252941224505
```

To find SiLK flow records that have a particular flow key hash, use **rwfilter(1)**:

```
$ yaf < vlan.pcap                                              \
  | rwipfix2silk --interface-values=vlan                       \
  | rwfilter --plugin=flowkey.so --flowkey=2148415270,15775704 \
       --pass=stdout -                                         \
  | rwcut --plugin=flowkey.so --fields=1-5,in,flowkey,stime    \
       --ipv6=ignore --timestamp=epoch --num-rec=9
          sIP|          dIP|sPort|dPort|pro|   in|   flowkey|         sTime|
  10.128.87.50|     10.0.0.4|32942|   80|  6|    2|2148415270|1252941224.465|
     10.0.0.4| 10.128.87.50|   80|32942|  6|    2|  15775704|1252941224.465|
  10.128.87.50|     10.0.0.4|32942|   80|  6|    2|2148415270|1252941224.505|
```

When using **rwfilter**, it is best to specify the flow hash key for both the forward and reverse records.

Use **rwuniq(1)** to check if records with the same flow key hash appear more than twice.

```
$ yaf < vlan.pcap                                                \
  | rwipfix2silk --interface-values=vlan                         \
  | rwuniq --plugin=flowkey.so --fields=flowkey --flows=3-
   flowkey|   Records|
```

Since no flow records are printed, the maximum number of times a flow key hash appears is 2.

ENVIRONMENT

SILK_PATH

This environment variable gives the root of the install tree. When searching for plug-ins, a SiLK application may use this environment variable. See the FILES section for details.

SILK_PLUGIN_DEBUG

When set to 1, the SiLK applications print status messages to the standard error as they attempt to find and open the *flowkey.so* plug-in. A typical invocation using this variable is:

```
env SILK_PLUGIN_DEBUG=1 rwcut --plugin=flowkey.so --version
```

FILES

${SILK_PATH}/lib64/silk/flowkey.so

${SILK_PATH}/lib64/flowkey.so

${SILK_PATH}/lib/silk/flowkey.so

${SILK_PATH}/lib/flowkey.so

/usr/local/lib64/silk/flowkey.so

/usr/local/lib64/flowkey.so

/usr/local/lib/silk/flowkey.so

/usr/local/lib/flowkey.so

Possible locations for the plug-in.

SEE ALSO

rwcut(1), **rwfilter(1)**, **rwgroup(1)**, **rwsort(1)**, **rwstats(1)**, **rwuniq(1)**, **rwipfix2silk(1)**, **rwflowpack(8)**, **flowcap(8)**, **sensor.conf(5)**, **silk(7)**, **yaf(1)**, **getFlowKeyHash(1)**, **tcpdump(1)**

NOTES

The **flowkey** plug-in was added in SiLK 3.15.0.

flowrate

SiLK plug-in providing payload and rate filters and fields

SYNOPSIS

```
rwfilter --plugin=flowrate.so [--payload-bytes=INTEGER_RANGE]
      [--payload-rate=DECIMAL_RANGE]
      [--bytes-per-second=DECIMAL_RANGE]
      [--packets-per-second=DECIMAL_RANGE]
      [--flowrate-zero-duration=MICROSECONDS] ...

rwcut --plugin=flowrate.so --fields=FIELDS
      [--flowrate-zero-duration=MICROSECONDS] ...

rwgroup --plugin=flowrate.so --fields=FIELDS
      [--flowrate-zero-duration=MICROSECONDS] ...

rwsort --plugin=flowrate.so --fields=FIELDS
      [--flowrate-zero-duration=MICROSECONDS] ...

rwstats --plugin=flowrate.so --fields=FIELDS --values=FIELDS
      [--flowrate-zero-duration=MICROSECONDS] ...

rwuniq --plugin=flowrate.so --fields=FIELDS --values=FIELDS
      [--flowrate-zero-duration=MICROSECONDS] ...
```

DESCRIPTION

When loaded into **rwfilter(1)**, the **flowrate** plug-in provides switches that can partition flows based on bytes of payload and/or on the rates of data transfer.

For **rwcut(1)**, **rwgroup(1)**, **rwsort(1)**, **rwstats(1)**, and **rwuniq(1)**, the **flowrate** plug-in provides fields that will print, sort flows by, and group flows by the bytes of payload, bytes-per-packet, bytes-per-second, packets-per-second, and bytes of payload per second. The **flowrate** plug-in also provides aggregate value fields in **rwstats** and **rwuniq**.

The payload byte count is determined by subtracting from the total byte count in the flow the bytes of overhead used by the packet headers. The payload calculation assumes minimal packet headers---that is, there are no options in the packets. For TCP, the switch assumes there are no TCP timestamps in the packets. Thus, the calculated payload will be the maximum possible bytes of payload. If the packet-overhead is larger than the reported number of bytes, the value is zero.

The various flow-rate quantities are determined by dividing the payload byte count, packet count, or byte count by the duration of the flow, giving the average rate across the flow. Flow records whose duration is zero create a problem when computing a flow-rate.

If a flow record's reported duration is zero, the count is divided by a value which defaults to 400 microseconds and may be specified by the **--flowrate-zero-duration** switch. The switch accepts a minimum of 1 microsecond. The smallest (non-zero) duration SiLK flow records support is 1 millisecond (1000 microseconds).

Prior to SiLK 3.16.0, the **flowrate** plug-in used a duration of 1 second (1000000 microseconds) when the reported duration was zero except when the rate was used as an aggregate value field in **rwstats** or **rwuniq**.

The **flowrate** plug-in must be explicitly loaded into an application via the **--plugin** switch. The reason for this is due to name clashes with existing switches and fields. For example, adding the **--packets-per-second** switch to **rwfilter** means any short-cutting of the current **--packets** switch will fail.

OPTIONS

The **flowrate** plug-in provides the following options to the indicated applications.

Common Switches

The following switch is available whenever the **flowrate** plug-in has been loaded into a supported application:

--flowrate-zero-duration=*MICROSECONDS*

When computing a rate and a flow record has a duration of zero, assume the duration is actually *MICROSECONDS* microseconds. The *MICROSECONDS* value must be one or greater. If this switch is not specified, a duration of 400 microseconds is used. The smallest non-zero duration SiLK flow records support is 1 millisecond (1000 microseconds). *Since SiLK 3.16.0.*

rwfilter Switches

When the **flowrate** plug-in has been loaded, the following set of partitioning switches are added to **rwfilter**. To pass the filter, the record must pass the test implied by each switch. The form of the argument to each switch is described below. The partitioning switches are:

--payload-bytes=*INTEGER_RANGE*

Check whether the payload byte count is within *INTEGER_RANGE*.

--payload-rate=*DECIMAL_RANGE*

Check whether the average number of payload bytes seen per second in the flow is within *DECIMAL_RANGE*.

--packets-per-second=*DECIMAL_RANGE*

Check whether the average number of packets per second in the flow is within *DECIMAL_RANGE*.

--bytes-per-second=*DECIMAL_RANGE*

Check whether the average number of bytes per second in the flow is within *DECIMAL_RANGE*.

An *INTEGER_RANGE* is a range of two non-negative integers, and a *DECIMAL_RANGE* is a range of two non-negative decimal values with accuracy up to 0.0001. The ranges are specified as two values separated by a hyphen, *MIN-MAX*; for example `1-500` or `5.0-10.031`. If a single value is given (e.g., `3.14`), the range consists of that single value. The upper limit of the range may be omitted, such as `1-`, in which case the upper limit is set to the maximum possible value.

rwcut, rwgroup, rwsort, rwstats, and rwuniq Switches

--fields=*FIELDS*

FIELDS refers to a list of fields to use for the operation. The **flowrate** plug-in adds the following fields for display, sorting, and grouping using the **rwcut(1)**, **rwgroup(1)**, **rwsort(1)**, **rwstats(1)**, and **rwuniq(1)** tools:

payload-bytes
Print, sort by, or group by the number of bytes of payload.

payload-rate
Print, sort by, or group by the bytes of payload seen per second.

pckts/sec
Print, sort by, or group by the packets seen per second.

bytes/sec
Print, sort by, or group by the bytes seen per second.

bytes/packet
Print, sort by, or group by the average number of bytes contained in each packet.

--values=*AGGREGATES*

The **flowrate** plug-in adds the following aggregate value fields to **rwstats** and **rwuniq**. *AGGREGATES* refers to a list of values to compute for each bin. To compute these values, **flowrate** maintains separate sums for the numerator and denominator while reading the records, then **flowrate** computes the ratio when the output is generated.

payload-bytes
Compute the approximate bytes of payload for records in this bin.

payload-rate
Compute the average bytes of payload seen per second for records in this bin.

pckts/sec
Compute the average packets seen per second for records in this bin,

bytes/sec
Compute the average bytes seen per second for records in this bin.

bytes/packet
Compute the average number of bytes contained in each packet for records in this bin.

ENVIRONMENT

SILK_PATH

This environment variable gives the root of the install tree. When searching for plug-ins, a SiLK application may use this environment variable. See the FILES section for details.

SILK_PLUGIN_DEBUG

When set to 1, the SiLK applications print status messages to the standard error as they attempt to find and open the *flowrate.so* plug-in. A typical invocation using this variable is:

```
env SILK_PLUGIN_DEBUG=1 rwcut --plugin=flowrate.so --version
```

FILES

${SILK_PATH}/lib64/silk/flowrate.so

${SILK_PATH}/lib64/flowrate.so

${SILK_PATH}/lib/silk/flowrate.so

${SILK_PATH}/lib/flowrate.so

/usr/local/lib64/silk/flowrate.so

/usr/local/lib64/flowrate.so

/usr/local/lib/silk/flowrate.so

/usr/local/lib/flowrate.so

Possible locations for the plug-in.

SEE ALSO

rwcut(1), **rwfilter(1)**, **rwgroup(1)**, **rwsort(1)**, **rwstats(1)**, **rwuniq(1)**, **silk(7)**

int-ext-fields

SiLK plug-in providing internal/external ip/port fields

SYNOPSIS

```
rwcut --plugin=int-ext-fields.so --fields=FIELDS ...

rwgroup --plugin=int-ext-fields.so --fields=FIELDS ...

rwsort --plugin=int-ext-fields.so --fields=FIELDS ...

rwstats --plugin=int-ext-fields.so --fields=FIELDS ...

rwuniq --plugin=int-ext-fields.so --fields=FIELDS ...
```

DESCRIPTION

The **int-ext-fields** plug-in adds four potential fields to **rwcut(1)**, **rwgroup(1)**, **rwsort(1)**, **rwstats(1)**, and **rwuniq(1)**. These fields contain the internal IP (`int-ip`), the external IP (`ext-ip`), the internal port (`int-port`, and the external port (`ext-port`). To use these fields, specify their names in the **--fields** switch.

These fields can be useful when a file contains flow records that were collected for multiple directions---for example, some flow records are incoming and some are outgoing.

For these fields to be available, the user must specify the list of flowtypes (i.e., class/type pairs) that are considered incoming and the list that are considered outgoing. The user must specify the flowtypes because SiLK has no innate sense of the direction of a flow record. Although "in" and "out" are common types, SiLK does not recognize that these represent flows going in opposite directions.

If a record has a flowtype that is not in the list of incoming and output flowtypes, the application uses a value of 0 for that field.

The user specifies the flowtypes by giving a comma-separated list of class/type pairs using the **--incoming-flowtypes** and **--outgoing-flowtypes** switches on the application's command line. When the switch is not provided, the application checks the INCOMING_FLOWTYPES and OUTGOING_FLOWTYPES environment variables. If the list of incoming and/or outgoing flowtypes are not specified, the fields are not available.

For the **packlogic-twoway(3)** site, one would set the following environment variables:

```
INCOMING_FLOWTYPES=all/in,all/inweb,all/inicmp,all/innull
OUTGOING_FLOWTYPES=all/out,all/outweb,all/outicmp,all/outnull
```

The parsing of flowtypes requires the **silk.conf(5)** site configuration file. You may need to set the SILK_CONFIG_FILE environment variable or specify **--site-config-file** on the command line prior to loading the plug-in.

OPTIONS

The **int-ext-fields** plug-in provides the following options to **rwcut**, **rwgroup**, **rwsort**, **rwstats**, and **rwuniq**.

--fields=*FIELDS*

FIELDS refers to a list of fields to use for the operation. The **int-ext-fields** plug-in adds the following fields for display, sorting, and grouping using the **rwcut(1)**, **rwgroup(1)**, **rwsort(1)**, **rwstats(1)**, and **rwuniq(1)** tools:

int-ip

Print, sort by, or group by the internal IP address. The internal IP is the destination address for incoming flowtypes and the source address for outgoing flowtypes. When a SiLK Flow record's flowtype is not listed in either the incoming or outgoing flowtypes list, the int-ip field is 0.

ext-ip

Print, sort by, or group by the external IP address. The external IP is the source address for incoming flowtypes and the destination address for outgoing flowtypes. When a SiLK Flow record's flowtype is not listed in either the incoming or outgoing flowtypes list, the ext-ip field is 0.

int-port

Print, sort by, or group by the internal port. This value is 0 for ICMP flow records, and when the SiLK Flow record's flowtype is not listed in either the incoming or outgoing flowtypes list.

ext-port

Print, sort by, or group by the external port. This value is 0 for ICMP flow records, and when the SiLK Flow record's flowtype is not listed in either the incoming or outgoing flowtypes list.

--incoming-flowtypes=*CLASS/TYPE*[,*CLASS/TYPE* ...]

Names the flowtypes that should be considered incoming. The list of flowtypes should be specified as a comma-separated list of class/type pairs. This switch overrides the flowtype list specified in the INCOMING_FLOWTYPES environment variable. If this switch is not provided and the INCOMING_FLOWTYPES environment variable is not set, the **int-ext-fields** plug-in will not define any fields.

--outgoing-flowtypes=*CLASS/TYPE*[,*CLASS/TYPE* ...]

Similar to **--incoming-flowtypes**, except it names the flowtypes that should be considered outgoing, and it overrides the OUTGOING_FLOWTYPES environment variable.

EXAMPLE

In the following example, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

Consider the file *data.rw* that contains data going in different directions:

```
$ rwcut --fields=sip,sport,dip,dport,proto,class,type data.rw
            sIP|sPort|            dIP|dPort|pro|cla|   type|
   10.239.86.13|29897|192.168.228.153|   25|  6|all|     in|
192.168.228.153|   25|   10.239.86.13|29897|  6|all|    out|
```

```
192.168.208.237|29416| 10.233.108.250|   25|  6|all|    out|
 10.233.108.250|   25|192.168.208.237|29416|  6|all|     in|
 192.168.255.94|29301|  10.198.18.193|   80|  6|all| outweb|
  10.198.18.193|   80| 192.168.255.94|29301|  6|all|  inweb|
   10.202.7.122|29438|192.168.248.202|   25|  6|all|     in|
192.168.248.202|   25|   10.202.7.122|29438|  6|all|    out|
 10.255.142.104|26731|192.168.236.220|   25|  6|all|     in|
192.168.236.220|   25| 10.255.142.104|26731|  6|all|    out|
```

Using the **int-ext-fields** plug-in allows one to print the internal and external addresses and ports (note: command line wrapped for improved readability):

```
$ rwcut --plugin=int-ext-fields.so                                  \
       --incoming=all/in,all/inweb --outgoing=all/out,all/outweb   \
       --fields=ext-ip,ext-port,int-ip,int-port,proto,class,type
         ext-ip|ext-p|         int-ip|int-p|pro|cla|   type|
   10.239.86.13|29897|192.168.228.153|   25|  6|all|     in|
   10.239.86.13|29897|192.168.228.153|   25|  6|all|    out|
 10.233.108.250|   25|192.168.208.237|29416|  6|all|    out|
 10.233.108.250|   25|192.168.208.237|29416|  6|all|     in|
  10.198.18.193|   80| 192.168.255.94|29301|  6|all| outweb|
  10.198.18.193|   80| 192.168.255.94|29301|  6|all|  inweb|
   10.202.7.122|29438|192.168.248.202|   25|  6|all|     in|
   10.202.7.122|29438|192.168.248.202|   25|  6|all|    out|
 10.255.142.104|26731|192.168.236.220|   25|  6|all|     in|
 10.255.142.104|26731|192.168.236.220|   25|  6|all|    out|
```

This can be especially useful when using a tool like **rwuniq** or **rwstats**:

```
$ export INCOMING_FLOWTYPES=all/in,all/inweb
$ export OUTGOING_FLOWTYPES=all/out,all/outweb
$ rwuniq --plugin=int-ext-fields.so               \
       --fields=int-ip,int-port --value=bytes
         int-ip|int-p|               Bytes|
192.168.208.237|29416|               28517|
192.168.248.202|   25|                4016|
192.168.228.153|   25|                3454|
192.168.236.220|   25|               31872|
 192.168.255.94|29301|               14147|
```

Beware of traffic whose type is not listed in INCOMING_FLOWTYPES or OUTGOING_FLOWTYPES

```
$ rwcut --num-rec=4 --fields=sip,sport,dip,dport,proto,type data2.rw
            sIP|sPort|            dIP|dPort|pro|   type|
     67.215.0.5|   53|      155.6.5.1| 1613| 17|ext2ext|
     67.215.0.5|   53|      155.6.5.1| 1895| 17|ext2ext|
     67.215.0.5|   53|      155.6.5.1| 1351| 17|ext2ext|
     67.215.0.5|   53|      155.6.5.1| 1988| 17|ext2ext|
```

since the **int-ext-fields** plug-in sets the fields to 0.

```
$ rwcut --num-rec=4 --plugin=int-ext-fields.so                    \
       --incoming=all/in,all/inweb --outgoing=all/out,all/outweb   \
       --fields=int-ip,int-port,ext-ip,ext-port,proto,type data4.rw
         int-ip|int-p|         ext-ip|ext-p|pro|   type|
        0.0.0.0|    0|        0.0.0.0|    0| 17|ext2ext|
        0.0.0.0|    0|        0.0.0.0|    0| 17|ext2ext|
        0.0.0.0|    0|        0.0.0.0|    0| 17|ext2ext|
        0.0.0.0|    0|        0.0.0.0|    0| 17|ext2ext|
```

ENVIRONMENT

INCOMING_FLOWTYPES

Used as the value for the **--incoming-flowtypes** when that switch is not provided.

OUTGOING_FLOWTYPES

Used as the value for the **--outgoing-flowtypes** when that switch is not provided.

SILK_CONFIG_FILE

This environment variable is used when the SiLK application attempts to locate the the SiLK site configuration file unless the **--site-config-file** switch is specified. Additional locations where the application searches are listed in the FILES section. The site configuration file is required to parse the flowtypes.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. As described in the FILES section, an application may use this environment variable when searching for the SiLK site configuration file.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files and plug-ins, an application may use this environment variable. See the FILES section for details.

SILK_PLUGIN_DEBUG

When set to 1, the SiLK applications print status messages to the standard error as they attempt to find and open the *int-ext-fields.so* plug-in. A typical invocation using this variable is

```
env SILK_PLUGIN_DEBUG=1 rwcut --plugin=int-ext-fields.so --version
```

FILES

${SILK_CONFIG_FILE}

${SILK_DATA_ROOTDIR}/silk.conf

/data/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided.

${SILK_PATH}/lib64/silk/int-ext-fields.so

${SILK_PATH}/lib64/int-ext-fields.so

${SILK_PATH}/lib/silk/int-ext-fields.so

${SILK_PATH}/lib/int-ext-fields.so

/usr/local/lib64/silk/int-ext-fields.so

/usr/local/lib64/int-ext-fields.so

/usr/local/lib/silk/int-ext-fields.so

/usr/local/lib/int-ext-fields.so

Possible locations for the plug-in.

SEE ALSO

rwcut(1), **rwgroup(1)**, **rwsort(1)**, **rwstats(1)**, **rwuniq(1)**, **silk.conf(5)**, **packlogic-twoway(3)**, **silk(7)**

BUGS

The `int-ip` and `ext-ip` fields do not respect the **--ip-format** switch nor SILK_IP_FORMAT environment variable. The IP addresses are printed in the canonical format, and the columns are wide enough for an IPv6 address.

ipafilter

SiLK plug-in for flow filtering based on IPA data

SYNOPSIS

```
rwfilter [--ipa-src-expr IPA_EXPR] [--ipa-dst-expr IPA_EXPR]
      [--ipa-any-expr IPA_EXPR] ...
```

DESCRIPTION

The **ipafilter** plug-in provides switches to **rwfilter(1)** that can partition flows using data in an IPA database. **rwfilter** will automatically load the **ipafilter** plug-in when it is available.

OPTIONS

The **ipafilter** plug-in provides the following options to rwfilter.

--ipa-src-expr=*IPA_EXPR*

Use *IPA_EXPR* to filter flows based on the source IP of the flow matching the *IPA_EXPR* expression.

--ipa-dst-expr=*IPA_EXPR*

Use *IPA_EXPR* to filter flows based on the destination IP of the flow matching the *IPA_EXPR* expression.

--ipa-any-expr=*IPA_EXPR*

Use *IPA_EXPR* to filter flows based on either the source or destination IP of the flow matching the *IPA_EXPR* expression.

IPA EXPRESSIONS

The syntax for IPA filter expressions is documented in **ipaquery(1)**. Some simple examples are shown in the **EXAMPLES** section below.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

To pull flows from or to any IP address in the "watch" catalog:

```
$ rwfilter --start-date 2010/01/01:00                  \
      --ipa-any-expr "in watch at 2010/01/01"          \
      --pass watchflows.rw
```

To pull flows from any IP labeled "bad" in the last year:

```
$ rwfilter --start-date 2010/01/01:00                  \
       --ipa-src-expr "label bad after 2009/01/01"     \
       --pass badguys.rw
```

ENVIRONMENT

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files and plug-ins, **rwfilter** may use this environment variable. See the FILES section for details.

SILK_PLUGIN_DEBUG

When set to 1, **rwfilter** prints status messages to the standard error as it attempts to find and open the *ipafilter.so* plug-in. A typical invocation using this variable is

```
env SILK_PLUGIN_DEBUG=1 rwfilter --plugin=ipafilter.so --version
```

FILES

$SILK_PATH/share/silk/silk-ipa.conf

$SILK_PATH/share/silk-ipa.conf

/usr/local/share/silk/silk-ipa.conf

/usr/local/share/silk-ipa.conf

Possible locations for the IPA configuration file. This file contains the URI for connecting to the IPA database. If the configuration file does not exist, attempts to use the **ipafilter** plug-in will exit with an error. The format of this URI is *driver*://*user*:*pass-word*@*hostname*/*database*. For example:

```
postgresql://ipauser:secret@database-server.domain.com/ipa
```

${SILK_PATH}/lib64/silk/ipafilter.so

${SILK_PATH}/lib64/ipafilter.so

${SILK_PATH}/lib/silk/ipafilter.so

${SILK_PATH}/lib/ipafilter.so

/usr/local/lib64/silk/ipafilter.so

/usr/local/lib64/ipafilter.so

/usr/local/lib/silk/ipafilter.so

/usr/local/lib/ipafilter.so

Possible locations for the plug-in.

SEE ALSO

rwfilter(1), **rwipaimport(1)**, **rwipaexport(1)**, **silk(7)**, **ipaquery(1)**, **ipaimport(1)**, **ipaexport(1)**

packlogic-generic.so

Packing logic for the **generic** site

SYNOPSIS

```
rwflowpack --packing-logic=packlogic-generic.so ...
```

DESCRIPTION

This manual page describes the *packlogic-generic.so* plug-in that defines the packing logic that **rwflowpack(8)** may use to categorize flow records. (This document uses the term *plug-in*, but the builder of SiLK may choose to compile the packing logic into **rwflowpack**. See the *SiLK Installation Handbook* for details.)

General Overview of rwflowpack

The primary job of **rwflowpack** is to categorize flow records into one or more *class* and *type* pairs. The class and type pair (also called a *flowtype*) are used by the analyst when selecting flow records from the data store using **rwfilter(1)**.

The settings that **rwflowpack** uses to categorize each flow record are determined by two textual configuration files and a compiled plug-in that is referred to as the *packing logic*.

The first of the configuration files is **silk.conf(5)** which specifies the classes, types, and sensors that **rwflowpack** uses when writing files and that **rwfilter** uses when selecting flow files.

The second configuration file is the **sensor.conf(5)** file. This file contains multiple **sensor** blocks, where each block contains information which the packing logic uses to categorize flow records collected by the probes specified for that sensor.

The combination of a *silk.conf* file and a particular packing logic plug-in define a *site*. By having the configuration and packing logic outside of the core tools, users can more easily configure SiLK for their particular installation and a single installation of SiLK can support multiple sites.

This manual page describes the packing logic for the **generic** site. For a description of the packing logic at another site, see that site's manual page.

- **packlogic-twoway(3)**

Networks, Classes, and Types for the "generic" Site

The *packlogic-generic.so* plug-in uses three *network* names to describe the logical address spaces that border the sensor:

internal

the space that is being monitored

external

the space outside the monitored network

null

the destination network for a flow that does not leave the router, because either the flow was blocked by the router's access control list or its destination was the router itself---e.g., a BGP message

The **generic** site assumes that all packets are either blocked by the sensor (that is, their destination is the **null** network), or that the packets cross the sensor so the source and destination networks always differ.

The packing logic also assumes that the above networks completely describe the space around the sensor. Since the **null** network is strictly a destination network, any flow that does not originate from the **external** network must originate from the **internal** network.

This allows the **generic** site to categorizes a flow record primarily by comparing a flow record's source to the **external** network, and the packing logic contains no comparisons to the **internal** network

The *silk.conf* file and *packlogic-generic.so* plug-in define a single class, **all**.

The type assigned to a flow record within the **all** class is one of:

in, inweb

Records whose source is the **external** network and whose destination is not the **null** network represent incoming traffic. The traffic is split into multiple types, and these types allow the analysts to query a subset of the flow records depending on their needs. Each incoming flow record is split into the one of incoming types using the following rules:

inweb

Contains traffic where the protocol is TCP (6) and either the source port or the destination port is one of 80, 443, or 8080

in

Contains all other incoming traffic.

out, outweb

Records whose source is not the **external** network and whose destination is not the **null** network represent outgoing traffic. The traffic is split among the types using rules similar to those for incoming traffic.

innull

Records whose source is the **external** network and whose destination is the **null** network represent blocked incoming traffic.

outnull

Records whose source is not the **external** network and whose destination is the **null** network represent blocked outgoing traffic.

Assigning a flow to source and destination networks

Since the **generic** site uses the **external** network to determine a flow record's type, each **sensor** block in the **sensor.conf(5)** file must specify a definition for the **external** network.

The *sensor.conf* file provides two ways to define a network: use the ***NET*-ipblocks** statement to specify the *NET* network as a list of IP address blocks, or use the ***NET*-interfaces** statement to specify the *NET* network using a list of SNMP interfaces.

For the source network of a flow record to be considered **external**, either the source IP (SiLK field `sIP`) must appear in the list of **external-ipblocks** or the incoming SNMP interface (SiLK field `in`) must appear in the list of **external-interfaces**. **Note:** If the probe block that specifies where the flow was collected contains an **interface-values vlan** statement, the SiLK `in` field contains the VLAN ID.

For the destination network of a flow record to be considered **null**, either the destination IP (`dIP`) must appear in the list of **null-ipblocks** or the outgoing SNMP interface (`out`) must appear in the list of **null-interfaces**.

Consider the following two sensors:

```
sensor S2
  ipfix-probes S2
  external-ipblocks 172.16.0.0/16
  internal-ipblocks 172.20.0.0/16
end sensor

sensor S3
  ipfix-probes S3
  external-interfaces 17,18,19
  internal-interfaces 21,22,23
end sensor
```

A flow record collected at probe S2 whose `sIP` is 172.16.1.1 is considered incoming, regardless of the destination IP.

A flow record collected at probe S3 whose `in` is 27 is considered outgoing. (Since `in` does not match the **external-interfaces**, the record is considered outgoing even though `in` does not match the **internal-interfaces** either.)

There are two constructs in the *sensor.conf* file that help when specifying these lists:

1. The ***NET*-interfaces** or ***NET*-ipblocks** statement in a **sensor** block may use **remainder** to denote interfaces or IP blocks that do not appear elsewhere in the block.
2. A **group** block can be used to give a name to a set of IP blocks or SNMP interfaces which a **sensor** block can reference.

For details, see the **sensor.conf(5)** manual page.

Valid sensors

When using the *packlogic-generic.so* plug-in, the **sensor** blocks in the *sensor.conf* file supports the following types of probes:

- **ipfix**
- **netflow-v5**
- **netflow-v9**

In addition, each **sensor** block must meet the following rules:

- Either **external-interfaces** or **external-ipblocks** must be specified. And,
- A sensor cannot mix ***NET*-ipblocks** and ***NET*-interfaces**, with the exception that **null-interfaces** are always allowed. And,
- Only one network on the sensor may use **remainder**. And,
- If a sensor contains only one ***NET*-ipblocks** statement, that statement may not use **remainder**. (The ***NET*-interfaces** statement does not have this restriction.)

Packing logic code

This section provides the logic used to assign the class and type at the **generic** site.

A single **sensor** block will assign the flow record to a single class and type, and processing of the flow for that **sensor** block stops as soon as a type is assigned. When multiple **sensor** blocks reference the same probe, the flow records collected by that probe are processed by each of those **sensor** blocks.

A flow record is always assigned to the class **all**.

A textual description of the code used to assign the type is shown here. As of SiLK 3.8.0, the type may be determined by the presence of certain IPFIX or NetFlowV9 information elements.

- If `sIP` matches **external-ipblocks** or `in` matches **external-interfaces**, then
  - If `dIP` matches **null-ipblocks** or `out` matches **null-interfaces**, pack as **innull**. Else,
  - Pack as **in** or **inweb**.
- If `dIP` matches **null-ipblocks** or `out` matches **null-interfaces**, pack as **outnull**. Else,
- Pack as **out** or **outweb**.
- Potentially modify the type: If the probe has a **quirks** setting that includes `firewall-event` and if the incoming record contains the `firewallEvent` or `NF_F_FW_EVENT` information element whose value is 3 (flow denied), change the type where the flow is packed as follows:
  - If the flow was denied due to an ingress ACL (`NF_F_FW_EXT_EVENT` of 1001), pack as **innull**.
  - If the flow was denied due to an egress ACL (`NF_F_FW_EXT_EVENT` of 1002), pack as **outnull**.
  - If the flow's current type is **innull**, **in**, or **inweb**, pack as **innull**.
  - If the flow's current type is **outnull**, **out**, or **outweb**, pack as **outnull**.

SEE ALSO

rwfilter(1), **rwflowpack(8)**, **sensor.conf(5)**, **silk.conf(5)**, **packlogic-twoway(3)**, **silk(7)**, *SiLK Installation Handbook*

packlogic-twoway.so

Packing logic for the **twoway** site

SYNOPSIS

```
rwflowpack --packing-logic=packlogic-twoway.so ...
```

DESCRIPTION

This manual page describes the *packlogic-twoway.so* plug-in that defines the packing logic that **rwflowpack(8)** may use to categorize flow records. (This document uses the term *plug-in*, but the builder of SiLK may choose to compile the packing logic into **rwflowpack**. See the *SiLK Installation Handbook* for details.)

General Overview of rwflowpack

The primary job of **rwflowpack** is to categorize flow records into one or more *class* and *type* pairs. The class and type pair (also called a *flowtype*) are used by the analyst when selecting flow records from the data store using **rwfilter(1)**.

The settings that **rwflowpack** uses to categorize each flow record are determined by two textual configuration files and a compiled plug-in that is referred to as the *packing logic*.

The first of the configuration files is **silk.conf(5)** which specifies the classes, types, and sensors that **rwflowpack** uses when writing files and that **rwfilter** uses when selecting flow files.

The second configuration file is the **sensor.conf(5)** file. This file contains multiple **sensor** blocks, where each block contains information which the packing logic uses to categorize flow records collected by the probes specified for that sensor.

The combination of a *silk.conf* file and a particular packing logic plug-in define a *site*. By having the configuration and packing logic outside of the core tools, users can more easily configure SiLK for their particular installation and a single installation of SiLK can support multiple sites.

This manual page describes the packing logic for the **twoway** site. For a description of the packing logic at another site, see that site's manual page.

- **packlogic-generic(3)**

Networks, Classes, and Types for the "twoway" Site

The *silk.conf* file and *packlogic-twoway.so* plug-in categorize a flow record based on how the packets that comprise the flow record moved between different *networks*.

The *packlogic-twoway.so* plug-in specifies three network names to describe the logical address spaces that border the sensor:

internal

the space that is being monitored

external

the space outside the monitored network

null

the destination network for a flow that does not leave the router, because either the flow was blocked by the router's access control list or its destination was the router itself---e.g., a BGP message

There is an implicit fourth network, **unknown**, which is anything that does not match the three networks above.

Given these networks, the following table describes how flows can move between the networks. Traffic between the networks is successfully routed unless the description explicitly says "blocked".

```
SOURCE    DESTINATION   DESCRIPTION
external  internal      incoming traffic
internal  external      outgoing traffic
external  null          blocked incoming traffic
internal  null          blocked outgoing traffic
external  external      strictly external traffic
internal  internal      strictly internal traffic
null      any           unclear: null should never be a source
external  unknown       unclear
internal  unknown       unclear
unknown   any           unclear
```

The *silk.conf* file and *packlogic-twoway.so* plug-in define a single class, **all**.

The type assigned to a flow record within the **all** class depends on the how the record moves between the networks, and the types follow from the table above:

in, inicmp, inweb

Incoming traffic. The traffic is split into multiple types, and these types allow the analysts to query a subset of the flow records depending on their needs. Each incoming flow record is split into the one of incoming types using the following rules:

inweb

Contains traffic where the protocol is TCP (6) and either the source port or the destination port is one of 80, 443, or 8080

inicmp

Contains flow records where either the protocol is ICMP (1) or the flow record is IPv6 and the protocol is ICMPV6 (58). By default, the **inicmp** and **outicmp** types are not used by the *packlogic-twoway.so* plug-in.

in

Contains all other incoming traffic.

out, outicmp, outweb

Outgoing traffic. The traffic is split among the types using rules similar to those for incoming traffic.

innull

Blocked incoming traffic

outnull

Blocked outgoing traffic

ext2ext

Strictly external traffic

int2int

Strictly internal traffic

other

Either traffic from the **null** network or traffic to or from the **unknown** network

Assigning a flow to source and destination networks

Each **sensor** block in the **sensor.conf(5)** file must specify how to determine the source and destination networks for each flow record collected by the probes specified for that sensor. There are two ways to do this.

The first method sets the source and destination of all records to particular networks. This can be used, for example, when the physical network device at the sensor only sees one direction of the traffic. To do this, use the **source-network** and **destination-network** statements in the **sensor** block. The following sensor, S1, considers all traffic as blocked incoming:

```
sensor S1
  ipfix-probes S1
  source-network external
  destination-network null
end sensor
```

The second method to determine how a flow record moves between the networks is to define the networks and use characteristics of the flow record to determine its source and destination networks.

The *sensor.conf* file provides two ways to define a network: use the ***NET*-ipblocks** statement to specify the *NET* network as a list of IP address blocks, or use the ***NET*-interfaces** statement to specify the *NET* network using a list of SNMP interfaces.

For the source network of a flow record to be considered **external**, either the source IP (SiLK field `sIP`) must appear in the list of **external-ipblocks** or the incoming SNMP interface (SiLK field `in`) must appear in the list of **external-interfaces**. **Note:** If the probe block that specifies where the flow was collected contains an **interface-values vlan** statement, the SiLK `in` field contains the VLAN ID.

For the destination network of a flow record to be considered **null**, either the destination IP (`dIP`) must appear in the list of **null-ipblocks** or the outgoing SNMP interface (`out`) must appear in the list of **null-interfaces**.

Consider the following two sensors:

```
sensor S2
  ipfix-probes S2
  external-ipblocks 172.16.0.0/16
  internal-ipblocks 172.20.0.0/16
end sensor
```

```
sensor S3
  ipfix-probes S3
  external-interfaces 17,18,19
  internal-interfaces 21,22,23
end sensor
```

A flow record collected at probe S2 whose `sIP` is 172.16.1.1 and whose `dIP` is 172.20.2.2 is considered incoming.

A flow record collected at probe S3 whose `in` is 23 and whose `out` is 18 is considered outgoing. A flow on S3 whose `in` is 23 and whose `out` is 27 is written to **other** since the `out` field is not matched.

There are two constructs in the *sensor.conf* file that help when specifying these lists:

1. The ***NET*-interfaces** or ***NET*-ipblocks** statement in a **sensor** block may use **remainder** to denote interfaces or IP blocks that do not appear elsewhere in the block.
2. A **group** block can be used to give a name to a set of IP blocks or SNMP interfaces which a **sensor** block can reference.

For details, see the **sensor.conf(5)** manual page.

Valid sensors

When using the *packlogic-twoway.so* plug-in, the **sensor** blocks in the *sensor.conf* file supports the following types of probes:

- **ipfix**
- **netflow-v5**
- **netflow-v9**
- **sflow**
- **silk**

In addition, each **sensor** block must meet the following rules:

- If the sensor has the **source-network** and **destination-network** explicitly set, the sensor is valid and none of the following checks are performed. Otherwise,
- At least one of ***NET*-interfaces** or ***NET*-ipblocks** must be specified, where *NET* is either **internal** or **external**. And,
- A sensor cannot mix ***NET*-ipblocks** and ***NET*-interfaces**, with the exception that **null-interfaces** are always allowed. And,
- Only one network on the sensor may use **remainder**. And,
- If a sensor contains only one ***NET*-ipblocks** statement, that statement may not use **remainder**. (The ***NET*-interfaces** statement does not have this restriction.) And,
- When the **remainder** keyword is not used and only one of the **internal** or **external** networks is defined, the **external** or **internal** network, respectively, is defined as having the **remainder**.

Packing logic code

This section provides the logic used to assign the class and type at the **twoway** site.

A single **sensor** block will assign the flow record to a single class and type, and processing of the flow for that **sensor** block stops as soon as a type is assigned. When multiple **sensor** blocks reference the same probe, the flow records collected by that probe are processed by each of those **sensor** blocks.

A flow record is always assigned to the class **all** unless the flow is ignored.

A textual description of the code used to assign the type is shown here. As of SiLK 3.8.0, the type may be determined by the presence of certain IPFIX or NetFlowV9 information elements.

- Ignore any flow record that matches a **discard-when** statement or does not match a **discard-unless** statement.
- If **source-network** is **external**, if `sIP` matches **external-ipblocks**, or if `in` matches **external-interfaces**, then
  - If **destination-network** is **null**, if `dIP` matches **null-ipblocks**, or if `out` matches **null-interfaces**, pack as **innull**. Else,
  - If **destination-network** is **internal**, if `dIP` matches **internal-ipblocks**, or if `out` matches **internal-interfaces**, pack as **in**, **inicmp**, or **inweb**. Else,
  - If **destination-network** is **external**, if `dIP` matches **external-ipblocks**, or if `out` matches **external-interfaces**, pack as **ext2ext**. Else,
  - Pack as **other**.
- Else, if **source-network** is **internal**, if `sIP` matches **internal-ipblocks**, or if `in` matches **internal-interfaces**, then
  - If **destination-network** is **null**, if `dIP` matches **null-ipblocks**, or if `out` matches **null-interfaces**, pack as **outnull**. Else,
  - If **destination-network** is **external**, if `dIP` matches **external-ipblocks**, or if `out` matches **external-interfaces**, pack as **out**, **outicmp**, or **outweb**. Else,
  - If **destination-network** is **internal**, if `dIP` matches **internal-ipblocks**, or if `out` matches **internal-interfaces**, pack as **int2int**. Else,
  - Pack as **other**.
- Else, pack as **other**.
- Potentially modify the type: If the probe has a **quirks** setting that includes `firewall-event` and if the incoming record contains the `firewallEvent` or `NF_F_FW_EVENT` information element whose value is 3 (flow denied), change the type where the flow is packed as follows:
  - If the flow was denied due to an ingress ACL (`NF_F_FW_EXT_EVENT` of 1001), pack as **innull**.
  - If the flow was denied due to an egress ACL (`NF_F_FW_EXT_EVENT` of 1002), pack as **outnull**.
  - If the flow's current type is **in**, **inweb**, **inicmp**, or **ext2ext**, pack as **innull**.
  - If the flow's current type is **out**, **outweb**, **outicmp**, or **int2int**, pack as **outnull**.
  - Else leave the type as is (**innull**, **outnull**, or **other**).

SEE ALSO

rwfilter(1), **rwflowpack(8)**, **sensor.conf(5)**, **silk.conf(5)**, **packlogic-generic(3)**, **silk(7)**, *SiLK Installation Handbook*

pmapfilter

User-defined labels for IPs and protocol/port pairs

SYNOPSIS

```
rwfilter --pmap-file=[MAPNAME:]FILENAME
      [--pmap-file=[MAPNAME:]FILENAME ...]
      [--pmap-src-MAPNAME=LABELS] [--pmap-dst-MAPNAME=LABELS]
      [--pmap-any-MAPNAME=LABELS] ...

rwcut --pmap-file=[MAPNAME:]FILENAME
      [--pmap-file=[MAPNAME:]FILENAME ...]
      --fields=FIELDS [--pmap-column-width=NUM]

rwgroup --pmap-file=[MAPNAME:]FILENAME
      [--pmap-file=[MAPNAME:]FILENAME ...]
      --id-fields=FIELDS

rwsort --pmap-file=[MAPNAME:]FILENAME
      [--pmap-file=[MAPNAME:]FILENAME ...]
      --fields=FIELDS

rwstats --pmap-file=[MAPNAME:]FILENAME
      [--pmap-file=[MAPNAME:]FILENAME ...]
      --fields=FIELDS [--pmap-column-width=NUM]

rwuniq --pmap-file=[MAPNAME:]FILENAME
      [--pmap-file=[MAPNAME:]FILENAME ...]
      --fields=FIELDS [--pmap-column-width=NUM]
```

DESCRIPTION

Prefix maps provide a mapping from values on a SiLK Flow record to string labels. The binary prefix map file is created from textual input with **rwpmapbuild**. See the **rwpmapbuild(1)** manual page for the syntax of input file. This manual page describes how to use a prefix map file to augment the features of some commonly used SiLK applications.

A prefix map file maps either an IP address or a protocol/port pair to a label. The **mode** statement in the input to **rwpmapbuild** determines whether the prefix map file is a mapping for IPs or for protocol/port pairs. To see the mode of an existing prefix map, use **rwpmapcat(1)** and specify **--output-type=type**.

When using the prefix map file as described in this manual page, one typically uses the prefix map's **map-name**. The **map-name** statement in the input to **rwpmapbuild** allows one to assign the map-name when creating the prefix map. To see the map-name of an existing prefix map, use **rwpmapcat --output-type=mapname**. To assign a map-name when loading a prefix map file, use the **--pmap-file** switch and specify the map-name you want to use, a colon, and the file name. A map-name provided to the **--pmap-file** switch overrides the map-name in the file (if one exists).

When using a prefix map in **rwfilter(1)**, the map-name is combined with the prefix **--pmap-src-**, **--pmap-dst-**, or **--pmap-any-** to create the partitioning switches. When using the prefix map to create fields in

rwcut(1), **rwgroup(1)**, **rwsort(1)**, **rwstats(1)**, and **rwuniq(1)**, the map-name must be combined with the prefix `src-` or `dst-` to create the field names.

The applications support using multiple prefix map files in a single invocation. When using multiple prefix map files, each file must have a unique map-name (or be assigned a unique map-name on the command line).

When a prefix map file does not contain a map-name and no map-name is provided on the command line, SiLK processes the prefix map in legacy mode. When in legacy mode, only one prefix map file may be used. See the LEGACY section for details.

Three types of prefix map files are currently implemented:

proto-port

Maps a protocol/port pair to a label.

IPv4-address

Maps an IPv4 address to a label. When used with IPv6 addresses, an IPv6 address in the ::ffff:0:0/96 prefix is converted to IPv4 and mapped to the label. Any other IPv6 address is mapped to the label `UNKNOWN`.

IPv6-address

Maps an IPv6 address to a label. When used with an IPv4 address, the IPv4 address is converted to IPv6, mapping the IPv4 address into the ::ffff:0:0/96 prefix.

For more information on constructing prefix map files, see the **rwpmapbuild(1)** documentation. To view the contents, type, or map-name of a prefix map file, use **rwpmapcat(1)**. To map textual input to the labels in a prefix map, use **rwpmaplookup(1)**.

OPTIONS

The **--pmap-file** switch is used to load the prefix map into the application. Use of the prefix map varies by application.

To use a prefix map within a supported application, one or more **--pmap-file** switches are required. Multiple **--pmap-file** switches are allowed as long as each prefix map is associated with a unique map-name. The switch has two forms:

--pmap-file=*MAPNAME*:*FILENAME*

FILENAME refers to a prefix map file generated using **rwpmapbuild**. *MAPNAME* is a name that may be used to refer to the fields or options specific to that prefix map. Specify *FILENAME* as `-` or `stdin` to read the prefix map from the standard input.

--pmap-file=*FILENAME*

When a *MAPNAME* is not specified explicitly as part of the argument, the prefix map file is checked to determine if a map-name was set when the prefix map was created (see **rwpmapbuild**). If so, that map-name is used. If not, the prefix map is processed in legacy mode for backward compatibility. See LEGACY below for more information. A prefix map's map-name is printed by the **rwfileinfo(1)** command or by specifying **--output-types=mapname** to **rwpmapcat**.

rwfilter Switches

When using the prefix map in **rwfilter(1)**, the map-name is combined with the prefix **--pmap-src-**, **--pmap-dst-**, or **--pmap-any-** to create the partitioning switches; that is, the switch name depends *in part* on the map-name of the prefix map.

--pmap-src-*map-name*=*LABELS*

If the prefix map associated with *map-name* is an IP prefix map, this matches records with a source address that maps to a label contained in the list of labels in *LABELS*.

If the prefix map associated with *map-name* is a proto-port prefix map, this matches records with a protocol and source port combination that maps to a label contained in the list of labels in *LABELS*.

--pmap-dst-*map-name*=*LABELS*

Similar to **--pmap-src-*map-name***, but uses the destination IP or the protocol and destination port.

--pmap-any-*map-name*=*LABELS*

If the prefix map associated with *map-name* is an IP prefix map, this matches records with a source or destination address that maps to a label contained in the list of labels in *LABELS*.

If the prefix map associated with *map-name* is a proto-port prefix map, this matches records with a protocol and a source or destination port combination that maps to a label contained in the list of labels in *LABELS*.

rwcut, rwgroup, rwsort, rwstats, and rwuniq Switches

When using the prefix map to create fields in **rwcut(1)**, **rwgroup(1)**, **rwsort(1)**, **rwstats(1)**, and **rwuniq(1)**, the map-name must be combined with the prefix `src-` or `dst-` to create the field names. The field names depend *in part* on the map-name of the prefix map.

--fields=*FIELDS*

FIELDS refers to a list of fields to use for the operation. Each prefix map associated with *map-name* creates two additional fields, `src-`*`map-name`* and `dst-`*`map-name`*, available for display, sorting, and counting using the **rwcut**, **rwgroup**, **rwsort**, **rwstats**, and **rwuniq** tools.

src-*map-name*

The value for the source from the prefix map file associated with *map-name*. For an IP-based prefix map file, this corresponds to the source IP. For a proto-port prefix map, it is the protocol/source-port.

dst-*map-name*

As **src-*map-name*** for the destination IP address or protocol/destination-port. It *is* possible to encode type and code in a proto-port prefix map, but it will only work when used for the protocol/destination-port.

--pmap-column-width=*NUM*

Set the maximum number of characters to use when displaying the textual value of any prefix map field in **rwcut**, **rwstats**, and **rwuniq** to *NUM*. This switch *must* precede the **--fields** switch. This switch is useful for prefix map files that have very long dictionary values.

LEGACY

When a prefix map file does not contain a map-name and no map-name is specified in the **--pmap-file** argument, SiLK processes the prefix map as it did prior to SiLK 2.0, which is called legacy mode. When in legacy mode, only one prefix map file may be used by the application. Legacy mode is deprecated, but it is maintained for backwards compatibility.

Legacy Switches

When a prefix map is loaded into **rwfilter** in legacy mode, the following switches are defined:

--pmap-saddress=*LABELS*

Match records with a source IP address that maps to a label contained in the list of labels in *LABELS*. Only works with IP prefix maps.

--pmap-daddress=*LABELS*

As **--pmap-saddress** for the destination IP.

--pmap-any-address=*LABELS*

Match records with a source or destination IP address that maps to a label contained in the list of labels in *LABELS*. Only works with IP prefix maps.

--pmap-sport-proto=*LABELS*

Match records with a protocol and source port combination that maps to a label contained in the list of labels in *LABELS*. Only works with proto-port prefix maps.

--pmap-dport-proto=*LABELS*

As **--pmap-saddress** for the protocol and destination port.

--pmap-any-port-proto=*LABELS*

Match records with a protocol and a source or destination port combination that maps to a label contained in the list of labels in *LABELS*. Only works with proto-port prefix maps.

Legacy Fields

When a prefix map is loaded into **rwcut**, **rwgroup**, **rwsort**, **rwstats**, or **rwuniq** in legacy mode, the following fields are made available to the **--fields** switch:

sval

The value from the prefix map file for the source. For an IP-based prefix map file, this corresponds to the source IP. For a proto-port prefix map, it is the protocol/source-port.

dval

As **sval** for the destination IP address or protocol/destination-port.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

The following examples explicitly specify the map name on the command line, ensuring the examples work any prefix map file. The examples use two prefix map files:

carnegiemellon.pmap

Maps the internal IP space of Carnegie Mellon to labels specifying the department that has been assigned that IP space. (An IPv4 prefix map provides a label for every IPv4 address; in this case, any IP outside of Carnegie Mellon's IP space is given the label `external`.)

service.pmap

Maps protocol/ports pairs to well-known services associated with those pairs (e.g., based the file */etc/protocols* and */etc/services*). For example, 80/tcp is labeled `TCP/HTTP`, 25/tcp is `TCP/SMTP`, ephemeral ports in protocol 6 are `TCP`, protocol 1 is `ICMP`, etc.

To find today's incoming flow records going to "FineArts":

```
$ rwfilter --type=in,inweb --pmap-file=CMU:carnegiemellon.pmap     \
       --pmap-dst-CMU="FineArts" --pass=fine-arts-in.rw
```

To find today's outgoing flow records coming from "ChemE":

```
$ rwfilter --type=out,outweb --pmap-file=CMU:carnegiemellon.pmap   \
       --pmap-src-CMU="ChemE" --pass=cheme-out.rw
```

To find today's internal traffic from "FineArts" to "ChemE":

```
$ rwfilter --type=int2int --pmap-file=CMU:carnegiemellon.pmap      \
       --pmap-src-CMU="FineArts" --pmap-dst-CMU="ChemE"            \
       --pass=finearts-to-cheme.rw
```

To find the reverse traffic:

```
$ rwfilter --type=int2int --pmap-file=CMU:carnegiemellon.pmap      \
       --pmap-src-CMU="ChemE" --pmap-dst-CMU="FineArts"            \
       --pass=cheme-to-finearts.rw
```

To find today's internal traffic that started or ended at "FineArts" and "ChemE" (this will find traffic between them, as well as traffic they had with any other university department):

```
$ rwfilter --type=int2int --pmap-file=CMU:carnegiemellon.pmap      \
       --pmap-any-CMU="ChemE,FineArts"                             \
       --pass=cheme-finearts.rw
```

Using the *service.pmap* file with **rwcut** to print the label for the protocol/port pairs:

```
$ rwcut --pmap-file=service:service.pmap                          \
       --fields=protocol,dport,dst-service,sport,src-service      \
       flow-records.rw
pro|dPort|dst-service|sPort|src-service|
 17|   53|    UDP/DNS|29617|        UDP|
 17|29617|        UDP|   53|    UDP/DNS|
  6|   22|    TCP/SSH|29618|        TCP|
  6|29618|        TCP|   22|    TCP/SSH|
  1|  771|       ICMP|    0|       ICMP|
 17|   67|   UDP/DHCP|   68|   UDP/DHCP|
  6|  443|  TCP/HTTPS|28816|        TCP|
  6|29897|        TCP|   25|   TCP/SMTP|
  6|29222|        TCP|   80|   TCP/HTTP|
 17|29361|        UDP|   53|    UDP/DNS|
```

Using the *service.pmap* file with **rwuniq**:

```
$ rwuniq --pmap-file=serv:service.pmap --fields=dst-serv          \
       --values=bytes flow-records.rw
   dst-serv|               Bytes|
    TCP/SSH|          3443906999|
   TCP/SMTP|           780000305|
        TCP|        114397570896|
  TCP/HTTPS|           387741258|
   TCP/HTTP|          1526975653|
    UDP/NTP|             1176632|
        UDP|            14404581|
   UDP/DHCP|             5121392|
    UDP/DNS|             3797474|
       ICMP|            10695328|
```

Using the *service.pmap* file with **rwstats**:

```
$ rwstats --pmap-file=srvc:service.pmap --fields=dst-srvc  \
       --values=bytes --count=5 flow-records.rw
INPUT: 501876 Records for 10 Bins and 120571390518 Total Bytes
OUTPUT: Top 5 Bins by Bytes
   dst-srvc|               Bytes|    %Bytes|   cumul_%|
        TCP|        114397570896| 94.879532| 94.879532|
    TCP/SSH|          3443906999|  2.856322| 97.735854|
   TCP/HTTP|          1526975653|  1.266449| 99.002303|
   TCP/SMTP|           780000305|  0.646920| 99.649223|
  TCP/HTTPS|           387741258|  0.321586| 99.970809|
```

Using **rwsort** with two prefix maps, where the records are first sorted by the originating department and then by the service they are requesting:

```
$ rwsort --pmap-file=service:service.pmap                 \
       --pmap-file=cmu:carnegiemellon.pmap                \
       --fields=src-cmu,dst-service flow-records.rw
```

To see the partitioning switches that a prefix map adds to **rwfilter**, load the prefix map file prior to specifying the **--help** switch.

```
$ rwfilter --pmap-file=carnegiemellon.pmap --help          \
  | sed -n '/^--pmap-/p'
```

To see the fields that a prefix map file adds to **rwcut**, **rwgroup**, **rwsort**, **rwstats**, or **rwuniq**, load the prefix map file prior to specifying **--help**, and then view the description of the **--fields** switch.

```
$ rwsort --pmap-file=service.pmap --help                   \
  | sed -n '/^--fields/,/^--/p'
```

SEE ALSO

rwcut(1), **rwfilter(1)**, **rwgroup(1)**, **rwpmapbuild(1)**, **rwpmapcat(1)**, **rwpmaplookup(1)**, **rwsort(1)**, **rwstats(1)**, **rwuniq(1)**, **rwfileinfo(1)**, **silk(7)**

PySiLK

Silk in Python

DESCRIPTION

This document describes the features of **PySiLK**, the SiLK Python extension. It documents the objects and methods that allow one to read, manipulate, and write SiLK Flow records, IPsets, Bags, and Prefix Maps (pmaps) from within **python(1)**. PySiLK may be used in a stand-alone Python script or as a plug-in from within the SiLK tools **rwfilter(1)**, **rwcut(1)**, **rwgroup(1)**, **rwsort(1)**, **rwstats(1)**, and **rwuniq(1)**. This document describes the objects and methods that PySiLK provides; the details of using those from within a plug-in are documented in the **silkpython(3)** manual page.

The SiLK Python extension defines the following objects and modules:

IPAddr object

Represents an IP Address.

IPv4Addr object

Represents an IPv4 Address.

IPv6Addr object

Represents an IPv6 Address.

IPWildcard object

Represents CIDR blocks or SiLK IP wildcard addresses.

IPSet object

Represents a SiLK IPset.

PrefixMap object

Represents a SiLK Prefix Map.

Bag object

Represents a SiLK Bag.

TCPFlags object

Represents TCP flags.

RWRec object

Represents a SiLK Flow record.

SilkFile object

Represents a channel for writing to or reading from SiLK Flow files.

FGlob object

Allows retrieval of filenames in a SiLK data store. See also the **silk.site** module.

silk.site module

Defines several functions that relate to the SiLK site configuration and allow iteration over the files in a SiLK data store.

silk.plugin module

Defines functions that may only be used in SiLK Python plug-ins.

The SiLK Python extension provides the following functions:

silk.get_configuration(*name*=None)

When *name* is **None**, return a dictionary whose keys specify aspects of how SiLK was compiled. When *name* is provided, return the dictionary value for that key, or **None** when *name* is an unknown key. The dictionary's keys and their meanings are:

COMPRESSION_METHODS

A list of strings specifying the compression methods that were compiled into this build of SiLK. The list will contain one or more of `NO_COMPRESSION`, `ZLIB`, `LZO1X`, and/or `SNAPPY`.

INITIAL_TCPFLAGS_ENABLED

True if SiLK was compiled with support for initial TCP flags; **False** otherwise.

IPV6_ENABLED

True if SiLK was compiled with IPv6 support; **False** otherwise.

SILK_VERSION

The version of SiLK linked with PySiLK, as a string.

TIMEZONE_SUPPORT

The string `UTC` if SiLK was compiled to use UTC, or the string `local` if SiLK was compiled to use the local timezone.

Since SiLK 3.8.1.

silk.ipv6_enabled()

Return **True** if SiLK was compiled with IPv6 support, **False** otherwise.

silk.initial_tcpflags_enabled()

Return **True** if SiLK was compiled with support for initial TCP flags, **False** otherwise.

silk.init_country_codes(*filename*=None)

Initialize PySiLK's country code database. *filename* should be the path to a country code prefix map, as created by **rwgeoip2ccmap(1)**. If *filename* is not supplied, SiLK will look first for the file specified by *$SILK_COUNTRY_CODES*, and then for a file named *country_codes.pmap* in *$SILK_PATH/share/silk*, *$SILK_PATH/share*, */usr/local/share/silk*, and */usr/local/share*. (The latter two assume that SiLK was installed in */usr/local*.) Will throw a **RuntimeError** if loading the country code prefix map fails.

silk.silk_version()

Return the version of SiLK linked with PySiLK, as a string.

IPAddr Object

An **IPAddr** object represents an IPv4 or IPv6 address. These two types of addresses are represented by two subclasses of **IPAddr**: **IPv4Addr** and **IPv6Addr**.

class silk.IPAddr(*address*)

The constructor takes a string *address*, which must be a string representation of either an IPv4 or IPv6 address, or an IPAddr object. IPv6 addresses are only accepted if silk.**ipv6_enabled()** returns **True**. The **IPAddr** object that the constructor returns will be either an **IPv4Addr** object or an **IPv6Addr** object.

For compatibility with releases prior to SiLK 2.2.0, the **IPAddr** constructor will also accept an integer *address*, in which case it converts that integer to an **IPv4Addr** object. This behavior is deprecated. Use the **IPv4Addr** and **IPv6Addr** constructors instead.

Examples:

```
>>> addr1 = IPAddr('192.160.1.1')
>>> addr2 = IPAddr('2001:db8::1428:57ab')
>>> addr3 = IPAddr('::ffff:12.34.56.78')
>>> addr4 = IPAddr(addr1)
>>> addr5 = IPAddr(addr2)
>>> addr6 = IPAddr(0x10000000) # Deprecated as of SiLK 2.2.0
```

Supported operations and methods:

Inequality Operations

In all the below inequality operations, whenever an IPv4 address is compared to an IPv6 address, the IPv4 address is converted to an IPv6 address before comparison. This means that **IPAddr("0.0.0.0") == IPAddr("::ffff:0.0.0.0")**.

addr1* == *addr2

Return **True** if *addr1* is equal to *addr2*; **False** otherwise.

addr1* != *addr2

Return **False** if *addr1* is equal to *addr2*; **True** otherwise.

addr1* < *addr2

Return **True** if *addr1* is less than *addr2*; **False** otherwise.

addr1* <= *addr2

Return **True** if *addr1* is less than or equal to *addr2*; **False** otherwise.

addr1* >= *addr2

Return **True** if *addr1* is greater than or equal to *addr2*; **False** otherwise.

addr1* > *addr2

Return **True** if *addr1* is greater than *addr2*; **False** otherwise.

***addr*.is_ipv6()**

Return **True** if *addr* is an IPv6 address, **False** otherwise.

***addr*.isipv6()**

(DEPRECATED in SiLK 2.2.0) An alias for **is_ipv6()**.

***addr*.to_ipv6()**

If *addr* is an **IPv6Addr**, return a copy of *addr*. Otherwise, return a new **IPv6Addr** mapping *addr* into the ::ffff:0:0/96 prefix.

***addr*.to_ipv4()**

If *addr* is an **IPv4Addr**, return a copy of *addr*. If *addr* is in the ::ffff:0:0/96 prefix, return a new **IPv4Addr** containing the IPv4 address. Otherwise, return **None**.

int(*addr*)

Return the integer representation of *addr*. For an IPv4 address, this is a 32-bit number. For an IPv6 address, this is a 128-bit number.

str(*addr*)

Return a human-readable representation of *addr* in its canonical form.

***addr*.padded()**

Return a human-readable representation of *addr* which is fully padded with zeroes. With IPv4, it will return a string of the form "xxx.xxx.xxx.xxx". With IPv6, it will return a string of the form "xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx".

***addr*.octets()**

Return a tuple of integers representing the octets of *addr*. The tuple's length is 4 for an IPv4 address and 16 for an IPv6 address.

***addr*.mask(*mask*)**

Return a copy of *addr* masked by the IPAddr *mask*.

When both addresses are either IPv4 or IPv6, applying the mask is straightforward.

If *addr* is IPv6 but *mask* is IPv4, *mask* is converted to IPv6 and then the mask is applied. This may result in an odd result.

If *addr* is IPv4 and *mask* is IPv6, *addr* will remain an IPv4 address if masking *mask* with `::ffff:0000:0000` results in `::ffff:0000:0000`, (namely, if bytes 10 and 11 of *mask* are 0xFFFF). Otherwise, *addr* is converted to an IPv6 address and the mask is performed in IPv6 space, which may result in an odd result.

***addr*.mask_prefix(*prefix*)**

Return a copy of *addr* masked by the high *prefix* bits. All bits below the *prefix*th bit will be set to zero. The maximum value for *prefix* is 32 for an **IPv4Addr**, and 128 for an **IPv6Addr**.

***addr*.country_code()**

Return the two character country code associated with *addr*. If no country code is associated with *addr*, return **None**. The country code association is initialized by the silk.**init_country_codes()** function. If **init_country_codes()** is not called before calling this method, it will act as if **init_country_codes()** was called with no argument.

IPv4Addr Object

An **IPv4Addr** object represents an IPv4 address. **IPv4Addr** is a subclass of **IPAddr**, and supports all operations and methods that **IPAddr** supports.

class silk.IPv4Addr(*address*)

The constructor takes a string *address*, which must be a string representation of IPv4 address, an **IPAddr** object, or an integer. A string will be parsed as an IPv4 address. An **IPv4Addr** object will be copied. An **IPv6Addr** object will be converted to an IPv4 address, or throw a **ValueError** if the conversion is not possible. A 32-bit integer will be converted to an IPv4 address.

Examples:

```
>>> addr1 = IPv4Addr('192.160.1.1')
>>> addr2 = IPv4Addr(IPAddr('::ffff:12.34.56.78'))
>>> addr3 = IPv4Addr(addr1)
>>> addr4 = IPv4Addr(0x10000000)
```

IPv6Addr Object

An **IPv6Addr** object represents an IPv6 address. **IPv6Addr** is a subclass of **IPAddr**, and supports all operations and methods that **IPAddr** supports.

class silk.IPv6Addr(*address*)

The constructor takes a string *address*, which must be a string representation of either an IPv6 address, an **IPAddr** object, or an integer. A string will be parsed as an IPv6 address. An **IPv6Addr** object will be copied. An **IPv4Addr** object will be converted to an IPv6 address. A 128-bit integer will be converted to an IPv6 address.

Examples:

```
>>> addr1 = IPAddr('2001:db8::1428:57ab')
>>> addr2 = IPv6Addr(IPAddr('192.160.1.1'))
>>> addr3 = IPv6Addr(addr1)
>>> addr4 = IPv6Addr(0x100000000000000000000000)
```

IPWildcard Object

An *IPWildcard* object represents a range or block of IP addresses. The *IPWildcard* object handles iteration over IP addresses with **for *x* in *wildcard***.

class silk.IPWildcard(*wildcard*)

The constructor takes a string representation *wildcard* of the wildcard address. The string *wildcard* can be an IP address, an IP with a CIDR notation, an integer, an integer with a CIDR designation, or an entry in SiLK wildcard notation. In SiLK wildcard notation, a wildcard is represented as an IP address in canonical form with each octet (IPv4) or hexadectet (IPv6) represented by one of following: a value, a range of values, a comma separated list of values and ranges, or the character 'x' used to represent the entire octet or hexadectet. IPv6 wildcard addresses are only accepted if silk.**ipv6_enabled()** returns **True**. The *wildcard* element can also be an IPWildcard, in which case a duplicate reference is returned.

Examples:

```
>>> a = IPWildcard('1.2.3.0/24')
>>> b = IPWildcard('ff80::/16')
>>> c = IPWildcard('1.2.3.4')
>>> d = IPWildcard('::ffff:0102:0304')
>>> e = IPWildcard('16909056')
>>> f = IPWildcard('16909056/24')
>>> g = IPWildcard('1.2.3.x')
>>> h = IPWildcard('1:2:3:4:5:6:7.x')
>>> i = IPWildcard('1.2,3.4,5.6,7')
>>> j = IPWildcard('1.2.3.0-255')
>>> k = IPWildcard('::2-4')
>>> l = IPWildcard('1-2:3-4:5-6:7-8:9-a:b-c:d-e:0-ffff')
>>> m = IPWildcard(a)
```

Supported operations and methods:

addr* in *wildcard

Return **True** if *addr* is in *wildcard*, **False** otherwise.

addr* not in *wildcard

Return **False** if *addr* is in *wildcard*, **True** otherwise.

string* in *wildcard

Return the result of **IPAddr(*string*) in *wildcard***.

string* not in *wildcard

Return the result of **IPAddr(*string*) not in *wildcard***.

***wildcard*.is_ipv6()**

Return **True** if *wildcard* contains IPv6 addresses, **False** otherwise.

str(*wildcard*)

Return the string that was used to construct *wildcard*.

IPSet Object

An **IPSet** object represents a set of IP addresses, as produced by **rwset(1)** and **rwsetbuild(1)**. The **IPSet** object handles iteration over IP addresses with **for *x* in *set***, and iteration over CIDR blocks using **for *x* in *set*.cidr_iter()**.

In the following documentation, and *ip_iterable* can be any of:

- an **IPAddr** object representing an IP address
- the string representation of a valid IP address
- an **IPWildcard** object
- the string representation of an **IPWildcard**
- an iterable of any combination of the above
- another **IPSet** object

class silk.IPSet(*[ip_iterable]*)

The constructor creates an empty IPset. If an *ip_iterable* is supplied as an argument, each member of *ip_iterable* will be added to the IPset.

Other constructors, all class methods:

silk.IPSet.load(*path*)

Create an **IPSet** by reading a SiLK IPset file. *path* must be a valid location of an IPset.

Other class methods:

silk.IPSet.supports_ipv6()

Return whether this implementation of IPsets supports IPv6 addresses.

Supported operations and methods:

In the lists of operations and methods below,

- *set* is an **IPSet** object
- *addr* can be an **IPAddr** object or the string representation of an IP address.
- *set2* is an **IPSet** object. The operator versions of the methods require an **IPSet** object.
- *ip_iterable* is an iterable over IP addresses as accepted by the **IPSet** constructor. Consider *ip_iterable* as creating a temporary **IPSet** to perform the requested method.

The following operations and methods do not modify the **IPSet**:

***set*.cardinality()**

Return the cardinality of *set*.

len(*set*)

Return the cardinality of *set*. In Python 2.x, this method will raise **OverflowError** if the number of IPs in the set cannot be represented by Python's Plain Integer type--that is, if the value is larger than `sys.maxint`. The **cardinality()** method will not raise this exception.

***set*.is_ipv6()**

Return **True** if *set* is a set of IPv6 addresses, and **False** if it a set of IPv4 addresses. For the purposes of this method, IPv4-in-IPv6 addresses (that is, addresses in the ::ffff:0:0/96 prefix) are considered IPv6 addresses.

addr* in *set

Return **True** if *addr* is a member of *set*; **False** otherwise.

addr* not in *set

Return **False** if *addr* is a member of *set*; **True** otherwise.

***set*.copy()**

Return a new **IPSet** with a copy of *set*.

***set*.issubset(*ip_iterable*)**

set* <= *set2

Return **True** if every IP address in *set* is also in *set2*. Return **False** otherwise.

***set*.issuperset(*ip_iterable*)**

set* >= *set2

Return **True** if every IP address in *set2* is also in *set*. Return **False** otherwise.

***set*.union(*ip_iterable*[, ...])**

***set* | *other* | ...**

Return a new **IPset** containing the IP addresses in *set* and all *other*s.

***set*.intersection(*ip_iterable*[, ...])**

***set* & *other* & ...**

Return a new **IPset** containing the IP addresses common to *set* and *other*s.

***set*.difference(*ip_iterable*[, ...])**

***set* - *other* - ...**

Return a new IPset containing the IP addresses in *set* but not in *other*s.

***set*.symmetric_difference(*ip_iterable*)**

set* ^ *other

Return a new IPset containing the IP addresses in either *set* or in *other* but not in both.

***set*.isdisjoint(*ip_iterable*)**

Return **True** when none of the IP addresses in *ip_iterable* are present in *set*. Return **False** otherwise.

***set*.cidr_iter()**

Return an iterator over the CIDR blocks in *set*. Each iteration returns a 2-tuple, the first element of which is the first IP address in the block, the second of which is the prefix length of the block. Can be used as **for (*addr*, *prefix*) in *set*.cidr_iter()**.

***set*.save(*filename*, *compression*=DEFAULT)**

Save the contents of *set* in the file *filename*. The *compression* determines the compression method used when outputting the file. Valid values are the same as those in silk.silkfile_open().

The following operations and methods **will** modify the **IPSet**:

***set*.add(*addr*)**

Add *addr* to *set* and return *set*. To add multiple IP addresses, use the **add_range()** or **update()** methods.

***set*.discard(*addr*)**

Remove *addr* from *set* if *addr* is present; do nothing if it is not. Return *set*. To discard multiple IP addresses, use the **difference_update()** method. See also the **remove()** method.

***set*.remove(*addr*)**

Similar to **discard()**, but raise **KeyError** if *addr* is not a member of *set*.

***set*.pop()**

Remove and return an arbitrary address from *set*. Raise **KeyError** if *set* is empty.

***set*.clear()**

Remove all IP addresses from *set* and return *set*.

***set*.convert(*version*)**

Convert *set* to an IPv4 IPset if *version* is 4 or to an IPv6 IPset if *version* is 6. Return *set*. Raise **ValueError** if *version* is not 4 or 6. If *version* is 4 and *set* contains IPv6 addresses outside of the ::ffff:0:0/96 prefix, raise **ValueError** and leave *set* unchanged.

***set*.add_range(*start*, *end*)**

Add all IP addresses between *start* and *end*, inclusive, to *set*. Raise **ValueError** if *end* is less than *start*.

***set*.update(*ip_iterable*[, ...])**

***set* |= *other* | ...**

Add the IP addresses specified in *other*s to *set*; the result is the union of *set* and *other*s.

***set*.intersection_update(*ip_iterable*[, ...])**

***set* &= *other* & ...**

Remove from *set* any IP address that does **not** appear in *other*s; the result is the intersection of *set* and *other*s.

***set*.difference_update(*ip_iterable*[, ...])**

***set* -= *other* | ...**

Remove from *set* any IP address found in *other*s; the result is the difference of *set* and *other*s.

***set*.symmetric_difference_update(*ip_iterable*)**

set* ^= *other

Update *set*, keeping the IP addresses found in *set* or in *other* but not in both.

RWRec Object

An **RWRec** object represents a SiLK Flow record.

class silk.RWRec(*[rec]*,*[field=value]*,...)

This constructor creates an empty **RWRec** object. If an **RWRec** *rec* is supplied, the constructor will create a copy of it. The variable *rec* can be a dictionary, such as that supplied by the **as_dict()** method. Initial values for record fields can be included.

Example:

```
>>> recA = RWRec(input=10, output=20)
>>> recB = RWRec(recA, output=30)
>>> (recA.input, recA.output)
(10, 20)
>>> (recB.input, recB.output)
(10, 30)
```

Instance attributes:

Accessing or setting attributes on an **RWRec** whose descriptions mention functions in the **silk.site** module causes the silk.site.**init_site()** function to be called with no argument if it has not yet been called successfully--that is, if silk.site.**have_site_config()** returns **False**.

***rec*.application**

The *service* port of the flow *rec* as set by the flow meter if the meter supports it, a 16-bit integer. The **yaf(1)** flow meter refers to this value as the *appLabel*. The default application value is 0.

***rec*.bytes**

The count of the number of bytes in the flow *rec*, a 32-bit integer. The default bytes value is 0.

***rec*.classname**

(READ ONLY) The class name assigned to the flow *rec*, a string. This value is first member of the tuple returned by the *rec.classtype* attribute, which see.

***rec*.classtype**

A 2-tuple containing the classname and the typename of the flow *rec*. Getting the value returns the result of §**??**. If that function throws an error, the result is a 2-tuple containing the string `?` and a string representation of *rec.classtype_id*. Setting the value to (*class*,*type*) sets *rec*.classtype_id to the result of §**??**. If that function throws an error because the (*class*,*type*) pair is unknown, *rec* is unchanged and **ValueError** is thrown.

***rec*.classtype_id**

The ID for the class and type of the flow *rec*, an 8-bit integer. The default classtype_id value is 255. Changes to this value are reflected in the *rec.classtype* attribute. The classtype_id attribute may be set to a value that is considered invalid by the **silk.site**.

***rec*.dip**

The destination IP of the flow *rec*, an **IPAddr** object. The default dip value is IPAddr('0.0.0.0'). May be set using a string containing a valid IP address.

***rec*.dport**

The destination port of the flow *rec*, a 16-bit integer. The default dport value is 0. Since the destination port field is also used to store the values for the ICMP type and code, setting this value may modify *rec*.icmptype and *rec*.icmpcode.

***rec*.duration**

The duration of the flow *rec*, a **datetime.timedelta** object. The default duration value is 0. Changing the *rec*.duration attribute will modify the *rec*.etime attribute such that (*rec*.etime - *rec*.stime) == the new *rec*.duration. The maximum possible duration is datetime.timedelta(milliseconds=0xffffffff). See also *rec*.duration_secs.

***rec*.duration_secs**

The duration of the flow *rec* in seconds, a float that includes fractional seconds. The default duration_secs value is 0. Changing the *rec*.duration_secs attribute will modify the *rec*.etime attribute in the same way as changing *rec*.duration. The maximum possible duration_secs value is 4294967.295.

***rec*.etime**

The end time of the flow *rec*, a **datetime.datetime** object. The default etime value is the UNIX epoch time, datetime.datetime(1970,1,1,0,0). Changing the *rec*.etime attribute modifies the flow record's duration. If the new duration would become negative or would become larger than **RWRec** supports, a **ValueError** will be raised. See also *rec*.etime_epoch_secs.

***rec*.etime_epoch_secs**

The end time of the flow *rec* as a number of seconds since the epoch time, a float that includes fractional seconds. Epoch time is 1970-01-01 00:00:00 UTC. The default etime_epoch_secs value is 0. Changing the *rec*.etime_epoch_secs attribute modifies the flow record's duration. If the new duration would become negative or would become larger than **RWRec** supports, a **ValueError** will be raised.

***rec*.initial_tcpflags**

The TCP flags on the first packet of the flow *rec*, a **TCPFlags** object. The default initial_tcpflags value is **None**. The *rec*.initial_tcpflags attribute may be set to a new **TCPFlags** object, or a string or number which can be converted to a **TCPFlags** object by the **TCPFlags()** constructor. Setting *rec*.initial_tcpflags when *rec*.session_tcpflags is **None** sets the latter to TCPFlags(''). Setting *rec*.initial_tcpflags or *rec*.session_tcpflags sets *rec*.tcpflags to the binary OR of their values. Trying to set *rec*.initial_tcpflags when *rec*.protocol is not 6 (TCP) will raise an **AttributeError**.

***rec*.icmpcode**

The ICMP code of the flow *rec*, an 8-bit integer. The default icmpcode value is 0. The value is only meaningful when *rec*.protocol is ICMP (1) or when ***rec*.is_ipv6()** is **True** and *rec*.protocol is ICMPv6 (58). Since a record's ICMP type and code are stored in the destination port, setting this value may modify *rec*.dport.

***rec*.icmptype**

The ICMP type of the flow *rec*, an 8-bit integer. The default icmptype value is 0. The value is only meaningful when *rec*.protocol is ICMP (1) or when ***rec*.is_ipv6()** is **True** and *rec*.protocol is ICMPv6 (58). Since a record's ICMP type and code are stored in the destination port, setting this value may modify *rec*.dport.

***rec*.input**

The SNMP interface where the flow *rec* entered the router or the vlanId if the packing tools are configured to capture it (see **sensor.conf(5)**), a 16-bit integer. The default input value is 0.

***rec*.nhip**

The next-hop IP of the flow *rec* as set by the router, an **IPAddr** object. The default nhip value is IPAddr('0.0.0.0'). May be set using a string containing a valid IP address.

***rec*.output**

The SNMP interface where the flow *rec* exited the router or the postVlanId if the packing tools are configured to capture it (see **sensor.conf(5)**), a 16-bit integer. The default output value is 0.

***rec*.packets**

The packet count for the flow *rec*, a 32-bit integer. The default packets value is 0.

***rec*.protocol**

The IP protocol of the flow *rec*, an 8-bit integer. The default protocol value is 0. Setting *rec*.protocol to a value other than 6 (TCP) causes *rec*.initial_tcpflags and *rec*.session_tcpflags to be set to **None**.

***rec*.sensor**

The name of the sensor where the flow *rec* was collected, a string. Getting the value returns the result of §**??**. If that function throws an error, the result is a string representation of *rec.sensor_id* or the string `?` when sensor_id is 65535. Setting the value to *sensor_name* sets *rec*.sensor_id to the result of §**??**. If that function throws an error because *sensor_name* is unknown, *rec* is unchanged and **ValueError** is thrown.

***rec*.sensor_id**

The ID of the sensor where the flow *rec* was collected, a 16-bit integer. The default sensor_id value is 65535. Changes to this value are reflected in the *rec.sensor* attribute. The sensor_id attribute may be set to a value that is considered invalid by **silk.site**.

***rec*.session_tcpflags**

The union of the flags of all but the first packet in the flow *rec*, a **TCPFlags** object. The default session_tcpflags value is **None**. The *rec*.session_tcpflags attribute may be set to a new **TCPFlags** object, or a string or number which can be converted to a **TCPFlags** object by the **TCPFlags()** constructor. Setting *rec*.session_tcpflags when *rec*.initial_tcpflags is **None** sets the latter to TCPFlags(''). Setting *rec*.initial_tcpflags or *rec*.session_tcpflags sets *rec*.tcpflags to the binary OR of their values. Trying to set *rec*.session_tcpflags when *rec*.protocol is not 6 (TCP) will raise an **AttributeError**.

***rec*.sip**

The source IP of the flow *rec*, an **IPAddr** object. The default sip value is IPAddr('0.0.0.0'). May be set using a string containing a valid IP address.

***rec*.sport**

The source port of the flow *rec*, an integer. The default sport value is 0.

***rec*.stime**

The start time of the flow *rec*, a **datetime.datetime** object. The default stime value is the UNIX epoch time, datetime.datetime(1970,1,1,0,0). Modifying the *rec*.stime attribute will modify the flow's end time such that *rec*.duration is constant. The maximum possible stime is 2038-01-19 03:14:07 UTC. See also *rec*.etime_epoch_secs.

***rec*.stime_epoch_secs**

The start time of the flow *rec* as a number of seconds since the epoch time, a float that includes fractional seconds. Epoch time is 1970-01-01 00:00:00 UTC. The default stime_epoch_secs value is 0. Changing the *rec*.stime_epoch_secs attribute will modify the flow's end time such that *rec*.duration is constant. The maximum possible stime_epoch_secs is 2147483647 (2^31-1).

***rec*.tcpflags**

The union of the TCP flags of all packets in the flow *rec*, a **TCPFlags** object. The default tcpflags value is TCPFlags(' '). The *rec*.tcpflags attribute may be set to a new **TCPFlags** object, or a string or number which can be converted to a **TCPFlags** object by the **TCPFlags()** constructor. Setting *rec*.tcpflags sets *rec*.initial_tcpflags and *rec*.session_tcpflags to **None**. Setting *rec*.initial_tcpflags or *rec*.session_tcpflags changes *rec*.tcpflags to the binary OR of their values.

***rec*.timeout_killed**

Whether the flow *rec* was closed early due to timeout by the collector, a boolean. The default timeout_killed value is **False**.

***rec*.timeout_started**

Whether the flow *rec* is a continuation from a timed-out flow, a boolean. The default timeout_started value is **False**.

***rec*.typename**

(READ ONLY) The type name of the flow *rec*, a string. This value is second member of the tuple returned by the *rec.classtype* attribute, which see.

***rec*.uniform_packets**

Whether the flow *rec* contained only packets of the same size, a boolean. The default uniform_packets value is **False**.

Supported operations and methods:

***rec*.is_icmp()**

Return **True** if the protocol of *rec* is 1 (ICMP) or if the protocol of *rec* is 58 (ICMPv6) and ***rec*.is_ipv6()** is **True**. Return **False** otherwise.

***rec*.is_ipv6()**

Return **True** if *rec* contains IPv6 addresses, **False** otherwise.

***rec*.is_web()**

Return **True** if *rec* can be represented as a web record, **False** otherwise. A record can be represented as a web record if the protocol is TCP (6) and either the source or destination port is one of 80, 443, or 8080.

***rec*.as_dict()**

Return a dictionary representing the contents of *rec*. Implicitly calls silk.site.**init_site()** with no arguments if silk.site.**have_site_config()** returns **False**.

***rec*.to_ipv4()**

Return a new copy of *rec* with the IP addresses (sip, dip, and nhip) converted to IPv4. If any of these addresses cannot be converted to IPv4, (that is, if any address is not in the ::ffff:0:0/96 prefix) return **None**.

***rec*.to_ipv6()**

Return a new copy of *rec* with the IP addresses (sip, dip, and nhip) converted to IPv6. Specifically, the function maps the IPv4 addresses into the ::ffff:0:0/96 prefix.

str(*rec*)

Return the string representation of ***rec*.as_dict()**.

rec1* == *rec2

Return **True** if *rec1* is structurally equivalent to *rec2*. Return **False** otherwise.

rec1* != *rec2

Return **True** if *rec1* is not structurally equivalent to *rec2* Return **False** otherwise.

SilkFile Object

A **SilkFile** object represents a channel for writing to or reading from SiLK Flow files. A SiLK file open for reading can be iterated over using **for *rec* in *file***.

Creation functions:

silk.silkfile_open(*filename*, *mode*, *compression*=DEFAULT, *notes*=[], *invocations*=[])

This function takes a filename, a mode, and a set of optional keyword parameters. It returns a **SilkFile** object. The *mode* should be one of the following constant values:

silk.READ
: Open file for reading

silk.WRITE
: Open file for writing

silk.APPEND

Open file for appending

The *filename* should be the path to the file to open. A few filenames are treated specially. The filename *stdin* maps to the standard input stream when the mode is **READ**. The filenames *stdout* and *stderr* map to the standard output and standard error streams respectively when the mode is **WRITE**. A filename consisting of a single hyphen (-) maps to the standard input if the mode is **READ**, and to the standard output if the mode is **WRITE**.

The *compression* parameter may be one of the following constants. (This list assumes SiLK was built with the required libraries. To check which compression methods are available at your site, see silk.**get_configuration("COMPRESSION_METHODS")**).

silk.DEFAULT

Use the default compression scheme compiled into SiLK.

silk.NO_COMPRESSION

Use no compression.

silk.ZLIB

Use zlib block compression (as used by **gzip(1)**).

silk.LZO1X

Use lzo1x block compression.

silk.SNAPPY

Use snappy block compression.

If *notes* or *invocations* are set, they should be list of strings. These add annotation and invocation headers to the file. These values are visible by the **rwfileinfo(1)** program.

Examples:

```
>>> myinputfile = silkfile_open('/path/to/file', READ)
>>> myoutputfile = silkfile_open('/path/to/file', WRITE,
                                 compression=LZO1X,
                                 notes=['My output file',
                                        'another annotation'])
```

silk.silkfile_fdopen(*fileno*, *mode*, *filename*=None, *compression*=DEFAULT, *notes*=[], *invocations*=[])

This function takes an integer file descriptor, a mode, and a set of optional keyword parameters. It returns a **SilkFile** object. The *filename* parameter is used to set the value of the *name* attribute of the resulting object. All other parameters work as described in the silk.**silkfile_open()** function.

Deprecated constructor:

class silk.SilkFile(*filename*, *mode*, *compression*=DEFAULT, *notes*=[], *invocations*=[])

This constructor creates a **SilkFile** object. The parameters are identical to those used by the **silkfile_open()** function. This constructor is deprecated as of SiLK 3.0.0. For future compatibility, please use the **silkfile_open()** function instead of the **SilkFile()** constructor to create **SilkFile** objects.

Instance attributes:

***file*.name**

The filename that was used to create *file*.

***file*.mode**

The mode that was used to create *file*. Valid values are **READ**, **WRITE**, or **APPEND**.

Instance methods:

***file*.read()**

Return an **RWRec** representing the next record in the **SilkFile** *file*. If there are no records left in the file, return **None**.

***file*.write(rec)**

Write the **RWRec** *rec* to the **SilkFile** *file*. Return **None**.

***file*.next()**

A **SilkFile** object is its own iterator. For example, **iter(*file*)** returns *file*. When the **SilkFile** is used as an iterator, the **next()** method is called repeatedly. This method returns the next record, or raises **StopIteration** once the end of file is reached

***file*.skip(*count*)**

Skip the next *count* records in *file* and return the number of records skipped. If the return value is less than *count*, the end of the file has been reached. At end of file, return 0. *Since SiLK 3.19.1.*

***file*.notes()**

Return the list of annotation headers for the file as a list of strings.

***file*.invocations()**

Return the list of invocation headers for the file as a list of strings.

***file*.close()**

Close the file and return **None**.

PrefixMap Object

A **PrefixMap** object represents an immutable mapping from IP addresses or protocol/port pairs to labels. **PrefixMap** objects are created from SiLK prefix map files as created by **rwpmapbuild(1)**.

class silk.PrefixMap(*filename*)

The constructor creates a prefix map initialized from the *filename*. The **PrefixMap** object will be of one of the two subtypes of **PrefixMap**: an **AddressPrefixMap** or a **ProtoPortPrefixMap**.

Supported operations and methods:

***pmap*[*key*]**

Return the string label associated with *key* in *pmap*. *key* must be of the correct type: either an **IPAddr** if *pmap* is an **AddressPrefixMap**, or a 2-tuple of integers (*protocol*, *port*), if *pmap* is a **ProtoPortPrefixMap**. The method raises **TypeError** when the type of the key is incorrect.

***pmap*.get(*key*, *default*=None)**

Return the string label associated with *key* in *pmap*. Return the value *default* if *key* is not in *pmap*, or if *key* is of the wrong type or value to be a key for *pmap*.

***pmap*.values()**

Return a tuple of the labels defined by the **PrefixMap** *pmap*.

***pmap*.iterranges()**

Return an iterator that will iterate over ranges of contiguous values with the same label. The return values of the iterator will be the 3-tuple (*start*, *end*, *label*), where *start* is the first element of the range, *end* is the last element of the range, and *label* is the label for that range.

Bag Object

A **Bag** object is a representation of a multiset. Each key represents a potential element in the set, and the key's value represents the number of times that key is in the set. As such, it is also a reasonable representation of a mapping from keys to integers.

Please note, however, that despite its set-like properties, **Bag** objects are not nearly as efficient as **IPSet** objects when representing large contiguous ranges of key data.

In PySiLK, the **Bag** object is designed to look and act similar to Python dictionary objects, and in many cases **Bag**s and **dict**s can be used interchangeably. There are differences, however, the primary of which is that **bag**[***key***] returns a value for all values in the key range of the bag. That value will be an integer zero for all key values that have not been incremented.

class silk.Bag(*mapping*=None, *key_type*=None, *key_len*=None, *counter_type*=None, *counter_len*=None)

The constructor creates a bag. All arguments are optional, and can be used as keyword arguments.

If *mapping* is included, the bag is initialized from that mapping. Valid mappings are:

- a **Bag**
- a key/value dictionary
- an iterable of key/value pairs

The *key_type* and *key_len* arguments describe the key field of the bag. The *key_type* should be a string from the list of valid types below. The *key_len* should be an integer describing the number of bytes that will represent values of *key_type*. The *key_type* argument is case-insensitive.

If *key_type* is not specified, it defaults to 'any-ipv6', unless silk.**ipv6_enabled()** is **False**, in which case the default is 'any-ipv4'. The one exception to this is when *key_type* is not specified, but *key_len* is specified with a value of less than 16. In this case, the default type is 'custom'.

Note: Key types that specify IPv6 addresses are not valid if silk.**ipv6_enabled()** returns **False**. An error will be thrown if they are used in this case.

If *key_len* is not specified, it defaults to the default number of bytes for the given *key_type* (which can be determined by the chart below). If specified, *key_len* must be one of the following integers: 1, 2, 4, 16.

The *counter_type* and *counter_len* arguments describe the counter value of the bag. The *counter_type* should be a string from the list of valid types below. The *counter_len* should be an integer describing the number of bytes that will represent valid of *counter_type*. The *counter_type* argument is case insensitive.

If *counter_type* is not specified, it defaults to 'custom'.

If *counter_len* is not specified, it defaults to 8. Currently, 8 is the only valid value of *counter_len*.

Here is the list of valid key and counter types, along with their default *key_len* values:

'sIPv4', 4

'dIPv4', 4

'sPort', 2

'dPort', 2

'protocol', 1

'packets', 4

'bytes', 4

'flags', 1

'sTime', 4

'duration', 4

'eTime', 4

'sensor', 2

'input', 2

'output', 2

'nhIPv4', 4

'initialFlags', 1

'sessionFlags', 1

'attributes', 1

'application', 2

'class', 1

'type', 1

'icmpTypeCode', 2

'sIPv6', 16

'dIPv6', 16

'nhIPv6', 16

'records', 4

'sum-packets', 4

'sum-bytes', 4

'sum-duration', 4

'any-ipv4', 4

'any-ipv6', 16

'any-port', 2

'any-snmp', 2

'any-time', 4

'custom', 4

Deprecation Notice: For compatibility with SiLK 2.x, the *key_type* argument may be a Python class. An object of the *key_type* class must be constructable from an integer, and it must possess an **__int__()** method which retrieves that integer from the object. Regardless of the maximum integer value supported by the *key_type* class, internally the bag will store the keys as type 'custom' with length 4.

Other constructors, all class methods:

silk.Bag.ipaddr(*mapping*, *counter_type*=None, *counter_len*=None)

Creates a **Bag** using 'any-ipv6' as the key type (or 'any-ipv4' if silk.**ipv6_enabled()** is **False**). *counter_type* and *counter_len* are used as in the standard **Bag** constructor. Equivalent to **Bag(***mapping***)**.

silk.Bag.integer(*mapping*, *key_len*=None, *counter_type*=None, *counter_len*=None)

Creates a **Bag** using 'custom' as the *key_type* (integer bag). *key_len*, *counter_type*, and *counter_len* are used as in the standard **Bag** constructor. Equivalent to **Bag(*mapping*, *key_type*='custom')**.

silk.Bag.load(*path*, *key_type*=None)

Creates a **Bag** by reading a SiLK bag file. *path* must be a valid location of a bag. When present, the *key_type* argument is used as in the **Bag** constructor, ignoring the key type specified in the bag file. When *key_type* is not provided and the bag file does not contain type information, the key is set to 'custom' with a length of 4.

silk.Bag.load_ipaddr(*path*)

Creates an IP address bag from a SiLK bag file. Equivalent to **Bag.load(*path*, *key_type* = IPv4Addr)**. This constructor is deprecated as of SiLK 3.2.0.

silk.Bag.load_integer(*path*)

Creates an integer bag from a SiLK bag file. Equivalent to **Bag.load(*path*, *key_type* = int)**. This constructor is deprecated as of SiLK 3.2.0.

Constants:

silk.BAG_COUNTER_MAX

This constant contains the maximum possible value for Bag counters.

Other class methods:

silk.Bag.field_types()

Returns a tuple of strings which are valid *key_type* or *counter_type* values.

silk.Bag.type_merge(*type_a*, *type_b*)

Given two types from **Bag.field_types()**, returns the type that would be given (by default) to a bag that is a result of the co-mingling of two bags of the given types. For example: **Bag.type_merge('sport','dport') == 'any-port'**.

Supported operations and methods:

In the lists of operations and methods below,

- *bag* and *bag2* are **Bag** objects
- *key* and *key2* are **IPAddr**s for bags that contain IP addresses, or integers for other bags
- *value* and *value2* are integers which represent the counter associated a key in the bag
- *ipset* is an **IPSet** object
- *ipwildcard* is an **IPWildcard** object

The following operations and methods do not modify the **Bag**:

***bag*.get_info()**

Return information about the keys and counters of the bag. The return value is a dictionary with the following keys and values:

'key_type'
The current key type, as a string.

'key_len'
The current key length in bytes.

'counter_type'
The current counter type, as a string.

'counter_len'
The current counter length in bytes.

The keys have the same names as the keyword arguments to the bag constructor. As a result, a bag with the same key and value information as an existing bag can be generated by using the following idiom: **Bag(*****bag*.**get_info())**.

***bag*.copy()**

Return a new **Bag** which is a copy of *bag*.

***bag*[*key*]**

Return the counter value associated with *key* in *bag*.

***bag*[*key*:*key2*] or *bag*[*key*,*key2*,...]**

Return a new **Bag** which contains only the elements in the key range [*key*, *key2*), or a new **Bag** containing only the given elements in the comma-separated list. In point of fact, the argument(s) in brackets can be any number of comma separated keys or key ranges. For example: ***bag*[1,5,15:18,20]** will return a bag which contains the elements 1, 5, 15, 16, 17, and 20 from *bag*.

***bag*[*ipset*]**

Return a new **Bag** which contains only elements in *bag* that are also contained in *ipset*. This is only valid for IP address bags. The *ipset* can be included as part of a comma-separated list of slices, as above.

***bag*[*ipwildcard*]**

Return a new **Bag** which contains only elements that are also contained in *ipwildcard*. This is only valid for IP address bags. The *ipwildcard* can be included as part of a comma-separated list of slices, as above.

key* in *bag

Return **True** if *bag*[*key*] is non-zero, **False** otherwise.

***bag*.get(*key*, *default*=None)**

Return *bag*[*key*] if *key* is in *bag*, otherwise return *default*.

***bag*.items()**

Return a list of **(*key*, *value*)** pairs for all keys in *bag* with non-zero values. This list is not guaranteed to be sorted in any order.

***bag*.iteritems()**

Return an iterator over **(*key*, *value*)** pairs for all keys in *bag* with non-zero values. This iterator is not guaranteed to iterate over items in any order.

***bag*.sorted_iter()**

Return an iterator over **(*key*, *value*)** pairs for all keys in *bag* with non-zero values. This iterator is guaranteed to iterate over items in key-sorted order.

***bag*.keys()**

Return a list of *key*s for all keys in *bag* with non-zero values. This list is guaranteed to be in key-sorted order.

***bag*.iterkeys()**

Return an iterkeys over *key*s for all keys in *bag* with non-zero values. This iterator is not guaranteed to iterate over keys in any order.

***bag*.values()**

Return a list of *value*s for all keys in *bag* with non-zero values. The list is guaranteed to be in key-sorted order.

***bag*.itervalues()**

Return an iterator over *value*s for all keys in *bag* with non-zero values. This iterator is not guaranteed iterate over values in any order, but the order is consistent with that returned by **iterkeys()**.

***bag*.group_iterator(*bag2*)**

Return an iterator over keys and values of a pair of **Bag**s. For each *key* which is in either *bag* or *bag2*, this iterator will return a **(*key*, *value*, *value2*)** triple, where *value* is *bag*.get(*key*), and *value2* is *bag*.get(*key*). This iterator is guaranteed to iterate over triples in *key* order.

bag* + *bag2

Add two bags together. Return a new **Bag** for which ***newbag*[*key*] = *bag*[*key*] + *bag2*[*key*]** for all keys in *bag* and *bag2*. Will raise an **OverflowError** if the resulting value for a key is greater than **BAG_COUNTER_MAX**. If the two bags are of different types, the resulting bag will be of a type determined by **Bag.type_merge()**.

bag* - *bag2

Subtract two bags. Return a new **Bag** for which ***newbag*[*key*] = *bag*[*key*] - *bag2*[*key*]** for all keys in *bag* and *bag2*, as long as the resulting value for that key would be non-negative. If the resulting value for a key would be negative, the value of that key will be zero. If the two bags are of different types, the resulting bag will be of a type determined by **Bag.type_merge()**.

***bag*.min(*bag2*)**

Return a new **Bag** for which ***newbag*[*key*] = min(*bag*[*key*], *bag2*[*key*])** for all keys in *bag* and *bag2*.

***bag*.max(*bag2*)**

Return a new **Bag** for which ***newbag*[*key*] = max(*bag*[*key*], *bag2*[*key*])** for all keys in *bag* and *bag2*.

***bag*.div(*bag2*)**

Divide two bags. Return a new **Bag** for which ***newbag*[*key*] = *bag*[*key*] / *bag2*[*key*])** rounded to the nearest integer for all keys in *bag* and *bag2*, as long as *bag2*[*key*] is non-zero. ***newbag*[*key*] = 0** when *bag2*[*key*] is zero. If the two bags are of different types, the resulting bag will be of a type determined by **Bag.type_merge()**.

bag* * *integer

integer* * *bag

Multiple a bag by a scalar. Return a new **Bag** for which ***newbag*[*key*] = *bag*[*key*] * *integer*** for all keys in *bag*.

***bag*.intersect(*set_like*)**

Return a new **Bag** which contains *bag*[*key*] for each *key* where ***key* in *set_like*** is true. *set_like* is any argument that supports Python's **in** operator, including **Bag**s, **IPSet**s, **IPWildcard**s, and Python sets, lists, tuples, et cetera.

***bag*.complement_intersect(*set_like*)**

Return a new **Bag** which contains *bag*[*key*] for each *key* where ***key* in *set_like*** is not true.

***bag*.ipset()**

Return an **IPSet** consisting of the set of IP address key values from *bag* with non-zero values. This only works if *bag* is an IP address bag.

***bag*.inversion()**

Return a new integer **Bag** for which all values from *bag* are inserted as key elements. Hence, if two keys in *bag* have a value of 5, *newbag*[5] will be equal to two.

bag* == *bag2

Return **True** if the contents of *bag* are equivalent to the contents of *bag2*, **False** otherwise.

bag* != *bag2

Return **False** if the contents of *bag* are equivalent to the contents of *bag2*, **True** otherwise.

***bag*.save(*filename*, *compression*=DEFAULT)**

Save the contents of *bag* in the file *filename*. The *compression* determines the compression method used when outputting the file. Valid values are the same as those in silk.silkfile_open().

The following operations and methods **will** modify the **Bag**:

***bag*.clear()**

Empty *bag*, such that *bag*[*key*] is zero for all keys.

bag*[*key*] = *value

Set the number of *key* in *bag* to *value*.

del *bag*[*key*]

Remove *key* from *bag*, such that *bag*[*key*] is zero.

***bag*.update(*mapping*)**

For each item in *mapping*, *bag* is modified such that for each key in *mapping*, the value for that key in *bag* will be set to the mapping's value. Valid mappings are those accepted by the **Bag()** constructor.

***bag*.add(*key*[, *key2*[, ...]])**

Add one of each *key* to *bag*. This is the same as incrementing the value for each *key* by one.

***bag*.add(*iterable*)**

Add one of each *key* in *iterable* to *bag*. This is the same as incrementing the value for each *key* by one.

***bag*.remove(*key*[, *key2*[, ...]])**

Remove one of each *key* from *bag*. This is the same as decrementing the value for each *key* by one.

***bag*.remove(*iterable*)**

Remove one of each *key* in *iterable* from *bag*. This is the same as decrementing the value for each *key* by one.

***bag*.incr(*key*, *value* = 1)**

Increment the number of *key* in *bag* by *value*. *value* defaults to one.

***bag*.decr(*key*, *value* = 1)**

Decrement the number of *key* in *bag* by *value*. *value* defaults to one.

bag* += *bag2

Equivalent to ***bag* = *bag* + *bag2***, unless an **OverflowError** is raised, in which case *bag* is no longer necessarily valid. When an error is not raised, this operation takes less memory than ***bag* = *bag* + *bag2***. This operation can change the type of *bag*, as determined by **Bag.type_merge()**.

bag* -= *bag2

Equivalent to ***bag* = *bag* - *bag2***. This operation takes less memory than ***bag* = *bag* - *bag2***. This operation can change the type of *bag*, as determined by **Bag.type_merge()**.

bag* *= *integer

Equivalent to ***bag* = *bag* * *integer***, unless an **OverflowError** is raised, in which case *bag* is no longer necessarily valid. When an error is not raised, this operation takes less memory than ***bag* = *bag* * *integer***.

***bag*.constrain_values(*min*=None, *max*=None)**

Remove *key* from *bag* if that key's value is less than *min* or greater than *max*. At least one of *min* or *max* must be specified.

***bag*.constrain_keys(*min*=None, *max*=None)**

Remove *key* from *bag* if that key is less than *min*, or greater than *max*. At least one of *min* or *max* must be specified.

TCPFlags Object

A **TCPFlags** object represents the eight bits of flags from a TCP session.

class silk.TCPFlags(*value*)

The constructor takes either a **TCPFlags** value, a string, or an integer. If a **TCPFlags** value, it returns a copy of that value. If an integer, the integer should represent the 8-bit representation of the flags. If a string, the string should consist of a concatenation of zero or more of the characters `F`, `S`, `R`, `P`, `A`, `U`, `E`, and `C`---upper or lower-case---representing the FIN, SYN, RST, PSH, ACK, URG, ECE, and CWR flags. As of SiLK 3.20.0, the constructor accepts `-` which sets all flags in the contructor. Spaces in the string are ignored.

Examples:

```
>>> a = TCPFlags('SA')
>>> b = TCPFlags(5)
```

Instance attributes (read-only):

***flags*.fin**

True if the FIN flag is set on *flags*, **False** otherwise

***flags*.syn**

True if the SYN flag is set on *flags*, **False** otherwise

***flags*.rst**

True if the RST flag is set on *flags*, **False** otherwise

***flags*.psh**

True if the PSH flag is set on *flags*, **False** otherwise

***flags*.ack**

True if the ACK flag is set on *flags*, **False** otherwise

***flags*.urg**

True if the URG flag is set on *flags*, **False** otherwise

***flags*.ece**

True if the ECE flag is set on *flags*, **False** otherwise

***flags*.cwr**

True if the CWR flag is set on *flags*, **False** otherwise

Supported operations and methods:

~flags

Return the bitwise inversion (not) of *flags*

flags1* & *flags2

Return the bitwise intersection (and) of the flags from *flags1* and *flags2*

flags1* | *flags2

Return the bitwise union (or) of the flags from *flags1* and *flags2*.

flags1* ^ *flags2

Return the bitwise exclusive disjunction (xor) of the flags from *flags1* and *flags2*.

int(*flags*)

Return the integer value of the flags set in *flags*.

str(*flags*)

Return a string representation of the flags set in *flags*.

***flags*.padded()**

Return a string representation of the flags set in *flags*. This representation will be padded with spaces such that flags will line up if printed above each other.

flags

When used in a setting that expects a boolean, return **True** if any flag value is set in *flags*. Return **False** otherwise.

***flags*.matches(*flagmask*)**

Given *flagmask*, a string of the form *high_flags*/*mask_flags*, return **True** if the flags of *flags* match *high_flags* after being masked with *mask_flags*; **False** otherwise. Given a *flagmask* without the slash (/), return **True** if all bits in *flagmask* are set in *flags*. I.e., a *flagmask* without a slash is interpreted as "*flagmask*/*flagmask*".

Constants:

The following constants are defined:

silk.TCP_FIN

A **TCPFlags** value with only the FIN flag set

silk.TCP_SYN

A **TCPFlags** value with only the SYN flag set

silk.TCP_RST

A **TCPFlags** value with only the RST flag set

silk.TCP_PSH

A **TCPFlags** value with only the PSH flag set

silk.TCP_ACK

A **TCPFlags** value with only the ACK flag set

silk.TCP_URG

A **TCPFlags** value with only the URG flag set

silk.TCP_ECE

A **TCPFlags** value with only the ECE flag set

silk.TCP_CWR

A **TCPFlags** value with only the CWR flag set

FGlob Object

An **FGlob** object is an iterable object which iterates over filenames from a SiLK data store. It does this internally by calling the **rwfglob(1)** program. The FGlob object assumes that the **rwfglob** program is in the PATH, and will raise an exception when used if not.

Note: It is generally better to use the silk.site.**repository_iter()** function from the silk.site Module instead of the **FGlob** object, as that function does not require the external **rwfglob** program. However, the **FGlob** constructor allows you to use a different site configuration file every time, whereas the silk.site.**init_site()** function only supports a single site configuration file.

class silk.FGlob(*classname*=None, *type*=None, *sensors*=None, *start_date*=None, *end_date*=None, *data_rootdir*=None, *site_config_file*=None)

Although all arguments have defaults, at least one of *classname*, *type*, *sensors*, *start_date* must be specified. The arguments are:

classname

if given, should be a string representing the class name. If not given, defaults based on the site configuration file, **silk.conf(5)**.

type

if given, can be either a string representing a type name or comma-separated list of type names, or can be a list of strings representing type names. If not given, defaults based on the site configuration file, *silk.conf*.

sensors

if given, should be either a string representing a comma-separated list of sensor names or IDs, and integer representing a sensor ID, or a list of strings or integers representing sensor names or IDs. If not given, defaults to all sensors.

start_date

if given, should be either a string in the format `YYYY/MM/DD[:HH]`, a date object, a datetime object (which will be used to the precision of one hour), or a time object (which is used for the given hour on the current date). If not given, defaults to start of current day.

end_date

if given, should be either a string in the format `YYYY/MM/DD[:HH]`, a date object, a datetime object (which will be used to the precision of one hour), or a time object (which is used for the given hour on the current date). If not given, defaults to *start_date*. The *end_date* cannot be specified without a *start_date*.

data_rootdir

if given, should be a string representing the directory in which to find the packed SiLK data files. If not given, defaults to the value in the SILK_DATA_ROOTDIR environment variable or the compiled-in default (/data).

site_config_file

if given, should be a string representing the path of the site configuration file, *silk.conf*. If not given, defaults to the value in the SILK_CONFIG_FILE environment variable or *$SILK_DATA_ROOTDIR/silk.conf*.

An **FGlob** object can be used as a standard iterator. For example:

```
for filename in FGlob(classname="all", start_date="2005/09/22"):
    for rec in silkfile_open(filename):
        ...
```

silk.site Module

The **silk.site** module contains functions that load the SiLK site file, and query information from that file.

silk.site.init_site(*siteconf*=None, *rootdir*=None)

Initializes the SiLK system's site configuration. The *siteconf* parameter, if given, should be the path and name of a SiLK site configuration file (see **silk.conf(5)**). If *siteconf* is omitted, the value specified in the environment variable SILK_CONFIG_FILE will be used as the name of the configuration file. If SILK_CONFIG_FILE is not set, the module looks for a file named *silk.conf* in the following directories: the directory specified by the *rootdir* argument, the directory specified in the SILK_DATA_ROOTDIR environment variable; the data root directory that is compiled into SiLK (/data); the directories *$SILK_PATH/share/silk/* and *$SILK_PATH/share/*.

The *rootdir* parameter, if given, should be the path to a SiLK data repository that a configuration that matches the SiLK site configuration. If *rootdir* is omitted, the value specified in the SILK_DATA_ROOTDIR environment variable will be used, or if that variable is not set, the data root directory that is compiled into SiLK (/data). The *rootdir* may be specified without a *siteconf* argument by using *rootdir* as a keyword argument. I.e., **init_site(rootdir="/data")**.

This function should not generally be called explicitly unless one wishes to use a non-default site configuration file.

The **init_site()** function can only be called successfully once. The return value of **init_site()** will be true if the site configuration was successful, or **False** if a site configuration file was not found. If a *siteconf* parameter was specified but not found, or if a site configuration file was found but did not parse properly, an exception will be raised instead. Once *init_site()* has been successfully invoked, silk.site.**have_site_config()** will return **True**, and subsequent invocations of **init_site()** will raise a **RuntimeError** exception.

Some **silk.site** methods and **RWRec** members require information from the *silk.conf* file, and when these methods are called or members accessed, the silk.site.**init_site()** function is implicitly invoked with no arguments if it has not yet been called successfully. The list of functions, methods, and attributes that exhibit this behavior include: silk.site.**sensors()**, silk.site.**classtypes()**, silk.site.**classes()**, silk.site.**types()**, silk.site.**default_types()**, silk.site.**default_class()**, silk.site.**class_sensors()**, silk.site.**sensor_id()**, silk.site.**sensor_from_id()**, silk.site.**classtype_id()**, silk.site.**classtype_from_id()**, silk.site.**set_data_rootdir()**, silk.site.**repository_iter()**, silk.site.**repository_silkfile_iter()**, silk.site.**repository_full_iter()**, ***rwrec*.as_dict()**, ***rwrec*.classname**, ***rwrec*.typename**, ***rwrec*.classtype**, and ***rwrec*.sensor**.

silk.site.have_site_config()

Return **True** if silk.site.**init_site()** has been called and was able to successfully find and load a SiLK configuration file, **False** otherwise.

silk.site.set_data_rootdir(*rootdir*)

Change the current SiLK data root directory once the *silk.conf* file has been loaded. This function can be used to change the directory used by the **silk.site** iterator functions. To change the SiLK data root directory before loading the *silk.conf* file, call silk.site.**init_site()** with a *rootdir* argument. **set_data_rootdir()** implicitly calls silk.site.**init_site()** with no arguments before changing the root directory if silk.site.**have_site_config()** returns **False**.

silk.site.get_site_config()

Return the current path to the SiLK site configuration file. Before silk.site.**init_site()** is called successfully, this will return the place that **init_site()** called with no arguments will first look for a configuration file. After **init_site()** has been successfully called, this will return the path to the file that **init_site()** loaded.

silk.site.get_data_rootdir()

Return the current SiLK data root directory.

silk.site.sensors()

Return a tuple of valid sensor names. Implicitly calls silk.site.**init_site()** with no arguments if silk.site.**have_site_config()** returns **False**. Returns an empty tuple if no site file is available.

silk.site.classes()

Return a tuple of valid class names. Implicitly calls silk.site.**init_site()** with no arguments if silk.site.**have_site_config()** returns **False**. Returns an empty tuple if no site file is available.

silk.site.types(*class*)

Return a tuple of valid type names for class *class*. Implicitly calls silk.site.**init_site()** with no arguments if silk.site.**have_site_config()** returns **False**. Throws **KeyError** if no site file is available or if *class* is not a valid class.

silk.site.classtypes()

Return a tuple of valid (class name, type name) tuples. Implicitly calls silk.site.**init_site()** with no arguments if silk.site.**have_site_config()** returns **False**. Returns an empty tuple if no site file is available.

silk.site.default_class()

Return the default class name. Implicitly calls silk.site.**init_site()** with no arguments if silk.site.**have_site_config()** returns **False**. Returns **None** if no site file is available.

silk.site.default_types(*class*)

Return a tuple of default types associated with class *class*. Implicitly calls silk.site.**init_site()** with no arguments if silk.site.**have_site_config()** returns **False**. Throws **KeyError** if no site file is available or if *class* is not a valid class.

silk.site.class_sensors(*class*)

Return a tuple of sensors that are in class *class*. Implicitly calls silk.site.**init_site()** with no arguments if silk.site.**have_site_config()** returns **False**. Throws **KeyError** if no site file is available or if *class* is not a valid class.

silk.site.sensor_classes(*sensor*)

Return a tuple of classes that are associated with *sensor*. Implicitly calls silk.site.**init_site()** with no arguments if silk.site.**have_site_config()** returns **False**. Throws **KeyError** if no site file is available or if *sensor* is not a valid sensor.

silk.site.sensor_description(*sensor*)

Return the sensor description as a string, or **None** if there is no description. Implicitly calls silk.site.**init_site()** with no arguments if silk.site.**have_site_config()** returns **False**. Throws **KeyError** if no site file is available or if *sensor* is not a valid sensor.

silk.site.sensor_id(*sensor*)

Return the numeric sensor ID associated with the string *sensor*. Implicitly calls silk.site.**init_site()** with no arguments if silk.site.**have_site_config()** returns **False**. Throws **KeyError** if no site file is available or if *sensor* is not a valid sensor.

silk.site.sensor_from_id(*id*)

Return the sensor name associated with the numeric sensor ID *id*. Implicitly calls silk.site.**init_site()** with no arguments if silk.site.**have_site_config()** returns **False**. Throws **KeyError** if no site file is available or if *id* is not a valid sensor identifier.

silk.site.classtype_id((*class*, *type*))

Return the numeric ID associated with the tuple (*class*, *type*). Implicitly calls silk.site.**init_site()** with no arguments if silk.site.**have_site_config()** returns **False**. Throws **KeyError** if no site file is available, if *class* is not a valid class, or if *type* is not a valid type in *class*.

silk.site.classtype_from_id(*id*)

Return the (*class*, *type*) name pair associated with the numeric ID *id*. Implicitly calls silk.site.**init_site()** with no arguments if silk.site.**have_site_config()** returns **False**. Throws **KeyError** if no site file is available or if *id* is not a valid identifier.

silk.site.repository_iter(*start*=None, *end*=None, *classname*=None, *types*=None, *classtypes*=None, *sensors*=None)

Return an iterator over file names in a SiLK repository. The repository is assumed to be in the data root directory that is returned by silk.site.**get_data_rootdir()** and to conform to the format of the current site configuration. This function implicitly calls silk.site.**init_site()** with no arguments if silk.site.**have_site_config()** returns **False**. See also silk.site.**repository_full_iter()** and silk.site.**repository_silkfile_iter()**.

The following types are accepted for *start* and *end*:

- a **datetime.datetime** object, which is considered to be specified to hour precision
- a **datetime.date** object, which is considered to be specified to day precision
- a string in the SiLK date format `YYYY/MM/DD[:HH]`, where the timezone depends on how SiLK was compiled; check the value of silk.**get_configuration("TIMEZONE_SUPPORT")**.

The rules for interpreting *start* and *end* are:

- When both *start* and *end* are specified to hour precision, files from all hours within that time range are returned.
- When *start* is specified to day precision, the hour specified in *end* (if any) is ignored, and files for all dates between midnight at *start* and the end of the day represented by *end* are returned.
- When *end* is not specified and *start* is specified to day precision, files for that complete day are returned.
- When *end* is not specified and *start* is specified to hour precision, files for that single hour are returned.
- When neither *start* nor *end* are specified, files for the current day are returned.
- It is an error to specify *end* without *start*, or to give an *end* that proceeds *start*.

To specify classes and types, either use the *classname* and *types* parameters or use the *classtypes* parameter. It is an error to use *classname* or *types* when *classtypes* is specified.

The *classname* parameter should be a named class that appears in silk.site.**classes()**. If neither *classname* nor *classtypes* are specified, *classname* will default to that returned by silk.site.**default_class()**.

The *types* parameter should be either a named type that appears in silk.site.**types(***classname***)** or a sequence of said named types. If neither *types* nor *classtypes* is specified, *types* will default to silk.site.**default_types(***classname***)**.

The *classtypes* parameter should be a sequence of (*classname*, *type*) pairs. These pairs must be in the sequence returned by silk.site.**classtypes()**.

The *sensors* parameter should be either a sensor name or a sequence of sensor names from the sequence returned by silk.site.**sensors()**. If *sensors* is left unspecified, it will default to the list of sensors supported by the given class(es).

silk.site.repository_silkfile_iter(*start*=None, *end*=None, *classname*=None, *types*=None, *classtypes*=None, *sensors*=None)

Works similarly to silk.site.**repository_iter()** except the file names that **repository_iter()** would return are opened as **SilkFile** objects and returned.

silk.site.repository_full_iter(*start*=None, *end*=None, *classname*=None, *types*=None, *classtypes*=None, *sensors*=None)

Works similarly to silk.site.**repository_iter()**. Unlike **repository_iter()**, this iterator's output will include the names of files that do not exist in the repository. The iterator returns (*filename*, *bool*) pairs where the *bool* value represents whether the given *filename* exists. For more information, see the description of the **--print-missing-files** switch in **rwfglob(1)**.

silk.plugin Module

silk.plugin is a module to support using PySiLK code as a plug-in to the **rwfilter(1)**, **rwcut(1)**, **rwgroup(1)**, **rwsort(1)**, **rwstats(1)**, and **rwuniq(1)** applications. The module defines the following methods, which are described in the **silkpython(3)** manual page:

silk.plugin.register_switch(*switch_name*, handler=*handler*, [arg=*needs_arg*], [help=*help_string*])

Define the command line switch **--*switch_name*** that can be used by the PySiLK plug-in.

silk.plugin.register_filter(*filter*, [finalize=*finalize*], [initialize=*initialize*])

Register the callback function *filter* that can be used by **rwfilter** to specify whether the flow record passes or fails.

silk.plugin.register_field(*field_name*, [add_rec_to_bin=*add_rec_to_bin*,] [bin_compare=*bin_compare*,] [bin_bytes=*bin_bytes*,] [bin_merge=*bin_merge*,] [bin_to_text=*bin_to_text*,] [column_width=*column_width*,] [description=*description*,] [initial_value=*initial_value*,] [initialize=*initialize*,] [rec_to_bin=*rec_to_bin*,] [rec_to_text=*rec_to_text*])

Define the new key field or aggregate value field named *field_name*. Key fields can be used in **rwcut**, **rwgroup**, **rwsort**, **rwstats**, and **rwuniq**. Aggregate value fields can be used in **rwstats** and **rwuniq**. Creating a field requires specifying one or more callback functions---the functions required depend on the application(s) where the field will be used. To simplify field creation for common field types, the remaining functions can be used instead.

silk.plugin.register_int_field(*field_name*, *int_function*, *min*, *max*, [*width*])

Create the key field *field_name* whose value is an unsigned integer.

silk.plugin.register_ipv4_field(*field_name*, *ipv4_function*, [*width*])

Create the key field *field_name* whose value is an IPv4 address.

silk.plugin.register_ip_field(*field_name*, *ipv4_function*, [*width*])

Create the key field *field_name* whose value is an IPv4 or IPv6 address.

silk.plugin.register_enum_field(*field_name*, *enum_function*, *width*, [*ordering*])

Create the key field *field_name* whose value is a Python object (often a string).

silk.plugin.register_int_sum_aggregator(*agg_value_name*, *int_function*, [*max_sum*], [*width*])

Create the aggregate value field *agg_value_name* that maintains a running sum as an unsigned integer.

silk.plugin.register_int_max_aggregator(*agg_value_name*, *int_function*, [*max_max*], [*width*])

Create the aggregate value field *agg_value_name* that maintains the maximum unsigned integer value.

silk.plugin.register_int_min_aggregator(*agg_value_name*, *int_function*, [*max_min*], [*width*])

Create the aggregate value field *agg_value_name* that maintains the minimum unsigned integer value.

EXAMPLE

Using PySiLK

The following is an example using the PySiLK bindings. The code is meant to show some standard PySiLK techniques, but is not otherwise meant to be useful.

The code reads each record in a SiLK flow file, checks whether the record's source port is 80/tcp or 8080/tcp and its volume is larger than 3 packets and 120 bytes, stores the destination IP of matching records in an IPset, and writes the IPset to a destination file. In addition, it prints the number of unique destination addresses and the addresses themselves to the standard output. Additional explanations can be found in-line in the comments.

```
#! /usr/bin/python

# Use print functions (Compatible with Python 3.0; Requires 2.6+)
from __future__ import print_function #Python2.6 or later required

# Import the PySiLK bindings
from silk import *

# Import sys for the command line arguments.
import sys

# Main function
def main():
    if len(sys.argv) != 3:
        print ("Usage: %s infile outset" % sys.argv[0])
        sys.exit(1)

    # Open a silk flow file for reading
    infile = silkfile_open(sys.argv[1], READ)

    # Create an empty IPset
    destset = IPSet()

    # Loop over the records in the file
    for rec in infile:

      # Do comparisons based on rwrec field values
      if (rec.protocol == 6 and rec.sport in [80, 8080] and
          rec.packets > 3 and rec.bytes > 120):
```

```
            # Add the dest IP of the record to the IPset
            destset.add(rec.dip)

    # Save the IPset for future use
    try:
        destset.save(sys.argv[2])
    except:
        sys.exit("Unable to write to %s" % sys.argv[2])

    # count the items in the set
    count = 0
    for addr in destset:
        count = count + 1

    print("%d addresses" % count)

    # Another way to do the same
    print("%d addresses" % len(destset))

    # Print the ip blocks in the set
    for base_prefix in destset.cidr_iter():
        print("%s/%d" % base_prefix)

# Call the main() function when this program is started
if __name__ == '__main__':
    main()
```

Adjusting the Class and Type Fields of a Flow File

Normally SiLK flow records get stamped with a class as flow records are recorded in the repository. However, if you are importing raw packet data or need to change some records that inadvertantly have the wrong class/type, PySiLK makes it easy to fix.

The example below sets the class to "all" and assigns a type of "in", "inweb", "out", or "outweb" to each record in an input file. The direction (in or out) is defined by an IPset that represents the internal network (traffic that neither comes from nor goes to the internal network is discarded in this example). Web/non-web flows are separated based on port.

```
#! /usr/bin/python

from __future__ import print_function #Python2.6 or later required
from silk import *
import silk.site
import sys                              # for command line args
from datetime import timedelta          # for date math

webports    = (80,443,8080)
inwebtype   = ("all","inweb")
intype      = ("all","in")
outwebtype  = ("all","outweb")
outtype     = ("all","out")
```

```
def main():
    if len(sys.argv) != 4:
        print("Usage:  %s infile setfile outfile" % sys.argv[0])
        sys.exit(1)

    #  open the SiLK file for reading
    infile = silkfile_open(sys.argv[1], READ)

    #  open the set file which represents my internal network
    #print(sys.argv[2])
    setfile = IPSet.load (sys.argv[2])

    # open the modified output file
    outfile = silkfile.open(sys.argv[3], WRITE)

    #  loop over the records in the file, shift time and write the update:
    for rec in infile:
        #
        #  If the src ip is in the set, it's going out.
        #  If the dst ip is in the set, it's coming in.
        #  If neither IP is in the set, discard the record.
        #
        if (rec.sport in webports) or (rec.dport in webports):
            if rec.sip in setfile:
                rec.classtype = outwebtype
                outfile.write(rec)
            elif rec.dip in setfile:
                rec.classtype = inwebtype
                outfile.write(rec)
        else:
            if rec.sip in setfile:
                rec.classtype = outtype
                outfile.write(rec)
            elif rec.dip in setfile:
                rec.classtype = intype
                outfile.write(rec)

    # clean up
    outfile.close()
    infile.close()

if __name__ == '__main__':
    main()
```

Changing Timestamps in a Flow File

On occasion you may find that you need to adjust all the timestamps for a SiLK flow file. For example, the flow file came from a packet capture file that was collected in a different time zone and had to be shifted a number of hours. Another possibility is if you need to adjust files because you determine the clock time was off.

It is relatively simple to change the timestamps using PySiLK. The sample code for changing data to another time zone is shown below; a minor change would shift the data by seconds instead of hours.

```
#! /usr/bin/python

from __future__ import print_function #Python2.6 or later required
from silk import *
import sys                              # for command line args
from datetime import timedelta          # for date math

def main():
    if len(sys.argv) != 4:
        print ("Usage:  %s infile offset-hours outfile" % sys.argv[0])
        sys.exit(1)

    #  open the SiLK file for reading
    infile = silkfile_open(sys.argv[1], READ)

    #  create the time offset object
    offset = timedelta(hours=int(sys.argv[2]))

    # open the modified output file
    outfile = silkfile_open(sys.argv[3], WRITE)

    #  loop over the records in the file, shift time and write the update:
    for rec in infile:
        rec.stime = rec.stime + offset
        outfile.write(rec)

    # clean up
    outfile.close()
    infile.close()

if __name__ == '__main__':
    main()
```

Grouping FTP Flow Records

The following script attempts to group all flows representing one direction of an FTP session and print them together. It takes as an argument the name of a file containing raw SiLK records sorted by start time and port number (`rwsort --fields=stime,sport`). The script extracts from the file all flows that potentially represent FTP traffic. We define a possible FTP flow as any flow where:

- the source port is 21 (FTP control channel)
- the source port is 20 (FTP data transfer port)
- both the source port and destination port are ephemeral (data transfer)

If a flow record has a source port of 21, the script adds the source and destination address to the list of possible FTP groups. The script categorizes each data transfer flow (source port 20 or ephemeral to ephemeral) according to its source and destination IP address pair. If a flow from the control channel with the same source and destination IP address exists the source and destination ports in the flow are added to the list of ports associated with the control channel interaction, otherwise the script lists the data transfer as being unclassified. After the entire file is processed, all FTP sessions that have been grouped are displayed.

```
#! /usr/bin/python

from __future__ import print_function #Python2.6 or later required
# import the necessary modules
import silk
import sys

# Test that the argument number is correct
if (len(sys.argv) != 2):
    print("Must supply a SiLK data file.")
    sys.exit()

# open the SiLK file for reading
rawFile=silk.silkfile_open(sys.argv[1], silk.READ)

# Initialize the record structure
# Unclassified will be the record ephemeral to ephemeral
# connections that don't appear to have a control channel
interactions = {"Unclassified":[]}

# Count of records processed
count = 0

# Process the input file
for rec in rawFile:
    count += 1
    key="%15s <--> %15s"%(rec.sip,rec.dip)
    if (rec.sport==21):
        if not key in interactions:
            interactions[key] = []
    else:
        if key in interactions:
            interactions[key].append("%5d <--> %5d"%(rec.sport,rec.dport))
        else:
            interactions["Unclassified"].append(
                "%15s:%5d <--> %15s:%5d"%(rec.sip,rec.sport,rec.dip,rec.dport))

# Print the count of all records
print(str(count) + " records processed")

# Print the groups of FTP flows
keyList = sorted(interactions.keys())
```

```
for key in keyList:
    print("\n" + key + " " + str(len(interactions[key])))
    if (key != "Unclassified"):
        for line in interactions[key]:
            print("   " + line)
```

Example output of the script:

```
184 records processed

xxx.xxx.xxx.236 <--> yyy.yyy.yyy.231 3
      20 <--> 56180
      20 <--> 56180
      20 <--> 58354

Unclassified 158
```

ENVIRONMENT

The following environment variables affect the tools in the SiLK tool suite.

SILK_CONFIG_FILE

This environment variable contains the location of the site configuration file, *silk.conf*. This variable will be used by silk.site.**init_site()** if no argument is passed to that method.

SILK_DATA_ROOTDIR

This variable gives the root of directory tree where the data store of SiLK Flow files is maintained, overriding the location that is compiled into the tools (/data). This variable will be used by the **FGlob** constructor unless an explicit **data_rootdir** value is specified. In addition, the silk.site.**init_site()** may search for the site configuration file, *silk.conf*, in this directory.

SILK_COUNTRY_CODES

This environment variable gives the location of the country code mapping file that the silk.**init_country_codes()** function will use when no name is given to that function. The value of this environment variable may be a complete path or a file relative to the SILK_PATH. See the FILES section for standard locations of this file.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value removes this restriction.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **PySiLK** may use this environment variable. See the FILES section for details.

PYTHONPATH

This is the search path that Python uses to find modules and extensions. The SiLK Python extension described in this document may be installed outside Python's installation tree; for example, in SiLK's installation tree. It may be necessary to set or modify the PYTHONPATH environment variable so Python can find the SiLK extension.

PYTHONVERBOSE

If the SiLK Python extension fails to load, setting this environment variable to a non-empty string may help you debug the issue.

SILK_PYTHON_TRACEBACK

When set, Python plug-ins (see **silkpython(3)**) will output trace back information regarding Python errors to the standard error.

PATH

This is the standard search path for executable programs. The **FGlob** constructor will invoke the **rwfglob(1)** program; the directory containing **rwfglob** should be included in the PATH.

TZ

When a SiLK installation is built to use the local timezone (to determine if this is the case, check the value of silk.**get_configuration("TIMEZONE_SUPPORT")**), the value of the TZ environment variable determines the timezone in which silk.site.**repository_iter()** parses timestamp strings. If the TZ environment variable is not set, the default timezone is used. Setting TZ to 0 or the empty string causes timestamps to be parsed as UTC. The value of the TZ environment variable is ignored when the SiLK installation uses utc. For system information on the TZ variable, see **tzset(3)**.

FILES

${SILK_CONFIG_FILE}

ROOT_DIRECTORY/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when no argument is passed to silk.site.**init_site()**.

${SILK_COUNTRY_CODES}

${SILK_PATH}/share/silk/country_codes.pmap

${SILK_PATH}/share/country_codes.pmap

/usr/local/share/silk/country_codes.pmap

/usr/local/share/country_codes.pmap

Possible locations for the country code mapping file used by silk.**init_country_codes()** when no name is given to the function.

${SILK_DATA_ROOTDIR}/

/data/

Locations for the root directory of the data repository. The silk.site.**init_site()** may search for the site configuration file, *silk.conf*, in this directory.

SEE ALSO

silkpython(3), **rwfglob(1)**, **rwfileinfo(1)**, **rwfilter(1)**, **rwcut(1)**, **rwpmapbuild(1)**, **rwset(1)**, **rwsetbuild(1)**, **rwgroup(1)**, **rwsort(1)**, **rwstats(1)**, **rwuniq(1)**, **rwgeoip2ccmap(1)**, **silk.conf(5)**, **sensor.conf(5)**, **silk(7)**, **python(1)**, **gzip(1)**, **yaf(1)**, **tzset(3)**, http://docs.python.org/

silk-plugin

Creating a SiLK run-time plug-in using C

SYNOPSIS

```
sk_cc=‘silk_config --compiler‘
sk_cflags=‘silk_config --cflags‘
$sk_cc $sk_cflags -shared -o FILENAME.so FILENAME.c

rwfilter --plugin=FILENAME.so [--plugin=FILENAME.so ...] ...

rwcut --plugin=FILENAME.so [--plugin=FILENAME.so ...]
      --fields=FIELDS ...

rwgroup --plugin=FILENAME.so [--plugin=FILENAME.so ...]
      --id-fields=FIELDS ...

rwsort --plugin=FILENAME.so [--plugin=FILENAME.so ...]
      --fields=FIELDS ...

rwstats --plugin=FILENAME.so [--plugin=FILENAME.so ...]
      --fields=FIELDS --values=VALUES ...

rwuniq --plugin=FILENAME.so [--plugin=FILENAME.so ...]
      --fields=FIELDS --values=VALUES ...
```

DESCRIPTION

Several of the SiLK analysis tools allow the user to augment the tools' functionality through the use of plug-ins that get loaded at run-time. These tools are:

rwfilter(1)

Supports adding new switches to determine whether each SiLK Flow record should be written in the **--pass** or the **--fail** output stream.

rwcut(1)

Supports adding new output fields that, when selected using the **--fields** switch, appear as a column in the output.

rwsort(1)

Supports adding new key fields that, when selected using the **--fields** switch, are used to determine the order in which records are sorted.

rwgroup(1)

Supports adding new key fields that, when selected using the **--id-fields** switch, are used to determine how records are grouped.

rwuniq(1)

Supports adding new key fields that, when selected using the **--fields** switch, are used to bin (i.e., group) the records. In addition, **rwuniq** supports adding new aggregate value fields that, when selected using the **--values** switch, will be computed for each bin. The key and value fields will appear in the output.

rwstats(1)

Supports adding new key fields that, when selected using the **--fields** switch, are used to bin (i.e., group) the records. In addition, **rwstats** supports adding new aggregate value fields that, when selected using the **--values** switch, will be computed for each bin and can be used to determine the top-N (or bottom-N) bins. The key and value fields will appear in the output for bins that meet the top-N threshold.

rwptoflow(1)

Supports adding functionality to ignore packets in the **pcap(3)** input stream or to modify the SiLK Flow records as the records are generated.

In addition, all of the above tools support adding new command line switches that can be used to initialize the plug-in itself (for example, to load an auxiliary file that the plug-in requires).

The plug-ins for all tools except **rwptoflow** can be written in either C or using PySiLK (the SiLK Python extension, see **pysilk(3)**). Although the execution time for PySiLK plug-ins is slower than for C plug-ins, we encourage you to use PySiLK for your plug-ins since the time-to-result can be faster for PySiLK: The faster development time in Python typically more than compensates for the slower execution time. Once you find that your PySiLK plug-in is seeing a great deal of use, or that PySiLK is just too slow for the amount of data you are processing, then re-write the plug-in using C. Even when you intend to write a plug-in using C, it can be helpful to prototype your plug-in using PySiLK.

The remainder of this document explains how to create a plug-in for the SiLK analysis tools (except **rwptoflow**) using the C programming language. For information on creating a plug-in using PySiLK, see **silkpython(3)**.

A template file for plug-ins is included in the SiLK source tree, in the *silk-VERSION/src/template/c-plugin.c* file.

The setup function

When you provide `--plugin=my-plugin.so` on the command line to an application, the application loads the **my-plugin.so** file and calls a setup function in that file to determine the new switches and/or fields that `my-plugin.so` provides.

This setup function is called with three arguments: the first two describe the version of the plug-in API, and the third is a pointer that is currently unused.

```
skplugin_err_t SKPLUGIN_SETUP_FN(
    uint16_t    major_version,
    uint16_t    minor_version,
    void       *plug_in_data)
{
    ...
}
```

There are several tasks this setup function may do: (1) check the API version, (2) register new command line switches (if any), (3) register new filters (if any), and (4) register new fields (if any). Let's describe these in more detail.

(1) Check the API version

The setup function should ensure that the plug-in and the application agree on the API to use. This provides protection in case the SiLK API to plug-ins changes in the future. To make this determination, call the **skpinSimpleCheckVersion()** function. A typical invocation is shown here, where the `major_version` and `minor_version` were passed into the **SKPLUGIN_SETUP_FN**, and `PLUGIN_API_VERSION_MAJOR` and `PLUGIN_API_VERSION_MINOR` are macros defined in the template file to the current version of the API.

```
#define PLUGIN_API_VERSION_MAJOR 1
#define PLUGIN_API_VERSION_MINOR 0


/* Check the plug-in API version */
rv = skpinSimpleCheckVersion(major_version, minor_version,
                             PLUGIN_API_VERSION_MAJOR,
                             PLUGIN_API_VERSION_MINOR,
                             skAppPrintErr);
if (rv != SKPLUGIN_OK) {
    return rv;
}
```

(2) Register command line switches

If the plug-in wants to define new command line switches, those switches must be registered in the setup function. A typical use of a command line switch is to allow the user to configure the plug-in; for example, the switch may allow the user to specify the location of an auxiliary input file that the plug-in requires, or to set a parameter used by the plug-in.

A second use for a command line switch is more subtle. When creating a plug-in for **rwfilter**, you may want your plug-in to provide several similar features, and only enable each feature when the user requests it via a command line switch. For this case, you want to delay registering the filter until the command line switch is seen, in which case the filter registration function should be invoked in the switch's callback function.

Information on registering a command line switch is available below (Registering command line switches).

(3) Register filters

You only need to register filters when the plug-in will be used by **rwfilter(1)**. You may choose to register the filters in the setup function; if you do, the filter will always be used when the plug-in is loaded by **rwfilter**. If you the plug-in provides several filtering functions that the user may choose from via command line switches, you should call the filter registration function in the callback function for the command line switch.

See Registering filter functions for details on registering a function to use with **rwfilter**.

(4) Register fields

If you want your plug-in to create a new printable field for **rwcut(1)**, a new sorting field for **rwsort(1)**, a new grouping field for **rwgroup(1)**, **rwstats(1)**, or **rwuniq(1)**, or a new aggregate value field for **rwstats** or **rwuniq**, you should register those fields in the setup function. (While you can register the fields in a switch's callback function, there is usually little reason to do so.)

There are two interfaces to registering a new field:

1. The advanced interface provides complete control over how the field is defined, and allows (or forces) you to specify exactly how to map from a SiLK Flow record to a binary representation to a textual representation. To use the advanced interface you will need to define several functions and fill in a C structure with pointers to those functions. This interface is described in the Advanced field registration function section below.

2. The simple interface can be used to define fields that map to an integer value, an IP address, or text that is index by an integer value. To use this interface, you need to define only one or two functions. The simple interface should handle many common cases, and it is described in Simple field registration functions.

Registering command line switches

When you register a switch, the two important pieces of information you must provide are a name for the switch and a callback function. When the application encounters the command line switch registered by your plug-in, the application will invoke the callback function with the parameter that the user provided (if any) to the command line switch.

To register a command line switch, call the **skpinRegOption2()** function:

```
skplugin_err_t skpinRegOption2(
    const char           *option_name,
    skplugin_arg_mode_t   mode,
    const char           *option_help_string,
    skplugin_help_fn_t    option_help_fn,
    skplugin_option_fn_t  opt_process_fn,
    void                 *opt_callback_data,
    int                   num_fn_mask,
    ...);  /* list of skplugin_fn_mask_t */
```

The parameters are

`option_name`

Specifies the command line switch to create. Do not include the leading `--` characters in the name.

`mode`

Determines whether the switch takes an argument. It should be one of

`NO_ARG`

when the command line option acts as an on/off switch

`OPTIONAL_ARG`

when the command line option has a default value, or

`REQUIRED_ARG`

when the user of the plug-in must provide an argument to the command line option.

`option_help_string`

This parameter specifies the usage string to print when the user requests **--help** from the application. This parameter may be NULL. Alternatively, you may instruct the application to generate a help string by invoking a callback function your plug-in provides, as described next.

`option_help_fn`

This parameter specifies a pointer to a function that the application will to call to print a help message for the command line switch when the user requests **--help** from the application. This parameter may be NULL; if it is not NULL, the `option_help_string` value is ignored. The signature of the function to provide is

```
void option_help_fn(
    FILE                *file_handle,
    const struct option *option,
    void                *opt_callback_data);
```

The `file_handle` argument is where the function should print its help message. The `opt_callback_data` is the value provided to **skpinRegOption2()** when the option was registered. The `struct option` parameter has two members of interest: `name` contains the number used to register the option, and `has_arg` contains the `mode` that was used when the option was specified.

`opt_process_fn`

Specifies the callback function, whose signature is

```
skplugin_err_t opt_process_fn(
    const char *opt_arg,
    void       *opt_callback_data);
```

The application will call `opt_process_fn(opt_arg,opt_callback_data)` when ***--option_name*** is seen as a command line argument. `opt_arg` will be the parameter the user passed to the switch, or it will be NULL if no parameter was given.

`opt_callback_data`

Will be passed back unchanged to the plug-in as a parameter in the **opt_process_fn()** and **option_help_fn()** callback functions.

`num_fn_mask`

Specifies the number of `skplugin_fn_mask_t` values specified as the final argument(s) to **skpinRegOption2()**.

`...`

Specifies a list of `skplugin_fn_mask_t` values. The length of this list must be specified in the `num_fn_mask` parameter. A plug-in file (e.g., *my-plugin.so*) can be loaded into any SiLK tool that supports plug-ins, but you may want a command line switch to appear only in certain applications. For example, the **flowrate(3)** plug-in can be used in both **rwfilter** and **rwcut**. When used by **rwfilter**, **flowrate** provides a **--bytes-per-second** switch; when used by **rwcut**, that switch is not available, and instead the **bytes/sec** field becomes available. This list determines in which applications the switch gets defined, and the list should contain the `SKPLUGIN_FN_*` or `SKPLUGIN_APP_*` macros defined in *skplugin.h*. To make the switch available in all applications, specify `SKPLUGIN_FN_ANY`. When **skpinRegOption2()** is called in an the application that does not match a value in this list, the function returns SKPLUGIN_ERR_DID_NOT_REGISTER, indicating that this option is not applicable to the application.

Registering filter functions

When you register a filter function, you are specifying a function that **rwfilter** will call for every SiLK Flow record that **rwfilter** reads from its input files. If the function returns `SKPLUGIN_FILTER_PASS`, **rwfilter** writes the record into the stream(s) specified by **--pass**. The record goes to the **--fail** streams if the function returns `SKPLUGIN_FILTER_FAIL`.

(The previous paragraph is true only when the plug-in is the only filtering predicate. When multiple tests are specified on the **rwfilter** command line, **rwfilter** will put the record into the fail destination as soon as any test fails. If there are multiple tests, your plug-in function will only see records that have not yet failed a test. If a plug-in filter function follows your function, it may fail a record that your filter function passed.)

To register a filter function, call the following function:

```
skplugin_err_t skpinRegFilter(
    skplugin_filter_t         **filter_handle,
    const skplugin_callbacks_t *regdata,
    void                       *cbdata);
```

`filter_handle`

When this parameter is not NULL, **skpinRegFilter()** will set the location it references to the newly created filter. Currently, no other function accepts the `skplugin_filter_t` as an argument.

`cbdata`

This parameter will be passed back unchanged to the plug-in as a parameter in the various callback functions. It may be NULL.

`regdata`

This structure has a member for every possible callback function the SiLK plug-in API supports. When used by **skpinRegFilter()**, the following members are supported.

`filter`

rwfilter invokes this function for each SiLK flow record. If the function returns SKPLUGIN_FILTER_PASS, the record is accepted; if it returns SKPLUGIN_FILTER_FAIL, the record is rejected. The type of the function is a `skplugin_filter_fn_t`, and its signature is:

```
skplugin_err_t filter(
    const rwRec  *rec,
    void         *cbdata,
    void        **extra);
```

where `rec` is the SiLK Flow record, `cbdata` is the `cbdata` specified in **skpinRegFilter()**, and `extra` will likely be unused.

`init`

rwfilter invokes this function for all registered filter predicates. It is called after argument processing and before reading records. The function's type is `skplugin_callback_fn_t` and the function pointer may be NULL. The callback's signature is

```
skplugin_err_t init(
    void *cbdata);
```

`cleanup`

When this function pointer is non-NULL, **rwfilter** calls this function after all records have been processed. This function has the same type and signature as the `init` function.

The function's return value will be `SKPLUGIN_OK` unless the `filter` member of the `regdata` structure is NULL.

If your plug-in registers a filter function and the plug-in is used in an application other that **rwfilter**, the call to **skpinRegFilter()** is a no-op.

Simple field registration functions

Using a plug-in, you can augment the keys available in the **--fields** switch on **rwcut(1)**, **rwgroup(1)**, **rwsort(1)**, **rwstats(1)**, and **rwuniq(1)**, and provide new aggregate value fields for the **--values** switch on **rwstats** and **rwuniq**.

The standard field registration function, **skpinRegField()**, is powerful---for example, you can control exactly how the value you compute will be printed. However, that power comes with complexity. Many times, all your plug-in needs to do is to compute a value, and having to write a function to print a number is work with little reward. The functions in this section handle the registration of common field types.

All of these functions require a name for the new field. The name is used as one of the arguments to the **--fields** or **--values** switch, and the name will also be used as the title when the field is printed (as in **rwcut**). Field names are case insensitive, and all field names must be unique within an application. You will get a run-time error if you attempt to create a field whose name already exists. (In **rwuniq** and **rwstats**, you may have a **--fields** key and a **--values** aggregate value with the same name.)

The callback functions dealing with integers use `uint64_t` for convenience, but internally the value will be stored in a smaller integer field if possible. Specifying the `max` parameter to the largest value you actually use may allow SiLK to use a smaller integer field.

The functions in this section return `SKPLUGIN_OK` unless the callback function is NULL.

Integer key field

The following function is used to register a key field whose value is an unsigned 64 bit integer.

```
skplugin_err_t skpinRegIntField(
    const char              *name,
    uint64_t                 min,
    uint64_t                 max,
    skplugin_int_field_fn_t  rec_to_int,
    size_t                   width);
```

`name`

The name of the new key field.

`min`

A number representing the minimum integer value for the field.

`max`

A number representing the maximum integer value for the field. If `max` is 0, a value of UINT64_MAX is used instead.

`rec_to_int`

A callback function that accepts a SiLK Flow record as its sole argument, and returns an unsigned integer (in host byte order) which represents the value of the `name` field for the given record. The signature is

```
uint64_t rec_to_int(
    const rwRec *rec);
```

`width`

The column width to use when displaying the field. If `width` is 0, it will be computed to be the number of digits necessary to display the integer `max`.

IPv4 key field

The following function registers a new key field whose value is an IPv4 address.

```
skplugin_err_t skpinRegIPv4Field(
    const char               *name,
    skplugin_ipv4_field_fn_t  rec_to_ipv4,
    size_t                    width);
```

`name`

The name of the new key field.

`rec_to_ipv4`

A callback function that accepts a SiLK Flow record as its sole argument, and returns a 32 bit integer (in host byte order) which represents the IPv4 addresses for the `name` field for the given record. The signature is

```
uint32_t rec_to_ipv4(
    const rwRec *rec);
```

`width`

The column width to use when displaying the field. If `width` is 0, it will be set to 15.

IP key field

The following function is used to register a key field whose value is any IP address (an `skipaddr_t`).

```
skplugin_err_t skpinRegIPAddressField(
    const char             *name,
    skplugin_ip_field_fn_t  rec_to_ipaddr,
    size_t                  width);
```

`name`

The name of the new key field.

`rec_to_ipaddr`

A callback function that accepts a SiLK Flow record and an `skipaddr_t` as arguments. The function should fill in the IP address as required for the `name` field. The signature is

```
void rec_to_ipaddr(
    skipaddr_t  *dest,
    const rwRec *rec);
```

`width`

The column width to use when displaying the field. If `width` is 0, it will be set to 39 when SiLK has support for IPv6 addresses, or 15 otherwise.

Text key field (from an integer)

The following function is used to register a key field whose value is an unsigned 64 bit integer (similar to **skpinRegIntField()**), but where the printed representation of the field is determined by a second callback function. This allows the plug-in to create arbitrary text for the field.

```
skplugin_err_t skpinRegTextField(
    const char               *name,
    uint64_t                  min,
    uint64_t                  max,
    skplugin_int_field_fn_t   value_fn,
    skplugin_text_field_fn_t  text_fn,
    size_t                    width);
```

`name`

The name of the new key field.

`min`

A number representing the minimum integer value for the field.

`max`

A number representing the maximum integer value for the field. If `max` is 0, a value of UINT64_MAX is used instead.

`value_fn`

A callback function that accepts a SiLK Flow record as its sole argument, and returns an unsigned integer (in host byte order) which represents the value of the `name` field for the given record. The signature is

```
uint64_t rec_to_int(
    const rwRec *rec);
```

`text_fn`

A callback function that provides the textual representation of the value returned by `value_fn`. The function's signature is

```
void text_fn(
    char     *dest,
    size_t    dest_len,
    uint64_t  val);
```

The callback should fill the character array `dest` with the printable representation of `val`. The number of characters in `dest` is given by `dest_len`. Note that `dest_len` may be different than the parameter `width` passed to **skpinRegTextField()**, and `text_fn` must NUL-terminate the string.

`width`

The column width to use when displaying the field.

Text key field (from a list)

The following function is used to register a field whose value is one of a list of strings. The plug-in provides the list of strings and a callback that takes a SiLK Flow record and returns an index into the list of strings.

```
skplugin_err_t skpinRegStringListField(
    const char               *name,
    const char              **list,
    size_t                    entries,
    const char               *default_value,
    skplugin_int_field_fn_t   rec_to_index,
    size_t                    width);
```

`name`

The name of the new key field.

`list`

List is the list of strings. The list should either be NULL terminated, or `entries` should have a non-zero value.

`entries`

The number of entries in `list`. If `entries` is 0, SiLK determines the number of entries by traversing `list` until it finds a element whose value is NULL.

`default_value`

The value to use when `rec_to_index` returns an invalid value.

`rec_to_index`

A callback function that accepts a SiLK Flow record as its sole argument, and returns an unsigned integer (in host byte order) which represents an index into `list`. If the return value is beyond the end of `list`, `default_value` will be used instead. The signature of this callback function is

```
uint64_t rec_to_int(
    const rwRec *rec);
```

`width`

The column width to use when displaying the field. If `width` is 0, it is defaulted to the width of the longest string in `list` and `default_value`.

Integer sum aggregate value field

The following function registers an aggregate value field that maintains a running unsigned integer sum. That is, the values returned by the callback are summed for every SiLK Flow record that matches a bin's key. The sum is printed when the bin is printed.

```
skplugin_err_t skpinRegIntSumAggregator(
    const char              *name,
    uint64_t                 max,
    skplugin_int_field_fn_t  rec_to_int,
    size_t                   width);
```

`name`

The name of the new aggregate value field.

`max`

A number representing the maximum integer value for the field. If `max` is 0, a value of UINT64_MAX is used instead.

`rec_to_int`

A callback function that accepts a SiLK Flow record as its sole argument, and returns an unsigned integer (in host byte order) which represents the value of the `name` value field for the given record. The signature is

```
uint64_t rec_to_int(
    const rwRec *rec);
```

`width`

The column width to use when displaying the value. If `width` is 0, it will be computed to be the number of digits necessary to display the integer `max`.

Integer minimum or maximum aggregate value field

The following function registers an aggregate value field that maintains the minimum integer value seen among all values returned by the callback function.

```
skplugin_err_t skpinRegIntMinAggregator(
    const char              *name,
    uint64_t                 max,
    skplugin_int_field_fn_t  rec_to_int,
    size_t                   width);
```

This function is similar, except it maintains the maximum value.

```
skplugin_err_t skpinRegIntMaxAggregator(
    const char              *name,
    uint64_t                 max,
    skplugin_int_field_fn_t  rec_to_int,
    size_t                   width);
```

`name`

The name of the new aggregate value field.

`max`

A number representing the maximum integer value for the field. If `max` is 0, a value of UINT64_MAX is used instead.

`rec_to_int`

A callback function that accepts a SiLK Flow record as its sole argument, and returns an unsigned integer (in host byte order) which represents the value of the `name` value field for the given record. The signature is

```
uint64_t rec_to_int(
    const rwRec *rec);
```

`width`

The column width to use when displaying the value. If `width` is 0, it will be computed to be the number of digits necessary to display the integer `max`.

Unsigned integer aggregate value field

The following function registers an aggregate value field that can be represented by a 64 bit integer. The plug-in must register two callback functions. The first takes a SiLK Flow record and returns an integer value; the second takes two integer values (as returned by the first callback function) and combines them to form a new aggregate value.

```
skplugin_err_t skpinRegIntAggregator(
    const char              *name,
    uint64_t                 max,
    skplugin_int_field_fn_t  rec_to_int,
    skplugin_agg_fn_t        agg,
    uint64_t                 initial,
    size_t                   width);
```

`name`

The name of the new aggregate value field.

`max`

A number representing the maximum integer value for the field. If `max` is 0, a value of UINT64_MAX is used instead.

`rec_to_int`

A callback function that accepts a SiLK Flow record as its sole argument, and returns an unsigned integer (in host byte order) which represents the value of the `name` value field for the given record. The signature is

```
uint64_t rec_to_int(
    const rwRec *rec);
```

`agg`

A callback function that combines (aggregates) two values. For example, if you wanted to create a new aggregate value that contained a bit-wise OR of the TCP flags seen on every packet, your `agg` function would OR the values. The signature is

```
uint64_t agg(
    uint64_t current,
    uint64_t operand);
```

`initial`

Specifies the initial value for the aggregate value. The first time the `agg` function is called on a bin, `operand` will be the value returned by `rec_to_int`, and `current` will be the value given in `initial`. The value in `initial` must be less than or equal to the value in `max`.

`width`

The column width to use when displaying the value. If `width` is 0, it will be computed to be the number of digits necessary to display the integer `max`.

Advanced field registration function

When the simple field registration functions do not provide what you need, you can use the **skpinRegField()** function that gives you complete control over the field.

skpinRegField() registers a new derived field for record processing. The plug-in must supply the name of the new field. The name is used as one of the arguments to the **--fields** switch (for key fields) or **--values** switch (for aggregate value fields). Field names are case insensitive, and all field names must be unique within an application. You will get a run-time error if you attempt to create a field whose name already exists. (In **rwuniq** and **rwstats**, you may have a **--fields** key and a **--values** aggregate value with the same name.)

The **skpinRegField()** function requires you initialize and pass in a structure. In this structure you will specify the callback functions that the application will call, as well as additional information required by some applications. Although the structure is complex, not all applications use all members.

If the plug-in is loaded by an application that does not support fields (such as **rwfilter**), the function is a no-op.

The advanced field registration function is

```
skplugin_err_t skpinRegField(
    skplugin_field_t          **return_field,
    const char                 *name,
    const char                 *description,
    const skplugin_callbacks_t *regdata,
    void                       *cbdata);
```

`return_field`

When this value is not NULL, **skpinRegField()** will set the location it references to the newly created field.

`name`

This sets the primary name of the field, and by default will be the title used when printing the field.

`description`

The `description` provides a textual description of the field. Currently this is unused.

`regdata`

The `regdata` structure provides the application with the callback functions and additional information it needs to use the plug-in. The members that must be set vary by application. It is described in more detail below.

`cbdata`

This parameter will be passed back unchanged to the plug-in as a parameter in the various callback functions. It may be NULL.

The structure used by the **skpinRegField()** (and **skpinRegFilter()**) functions to specify callback functions is shown here:

```
typedef struct skplugin_callbacks_st {
    skplugin_callback_fn_t     init;
    skplugin_callback_fn_t     cleanup;
```

```
    size_t                     column_width;
    size_t                     bin_bytes;
    skplugin_text_fn_t         rec_to_text;
    skplugin_bin_fn_t          rec_to_bin;
    skplugin_bin_fn_t          add_rec_to_bin;
    skplugin_bin_to_text_fn_t  bin_to_text;
    skplugin_bin_merge_fn_t    bin_merge;
    skplugin_bin_cmp_fn_t      bin_compare;
    skplugin_filter_fn_t       filter;
    skplugin_transform_fn_t    transform;
    const uint8_t             *initial;
    const char               **extra;
} skplugin_callbacks_t;
```

All of the callback functions reference in this structure take `cbdata` as a parameter, which is the value that was specified in the call to **skpinRegField()**. The `extra` parameter to the callback functions is used in complex plug-ins and can be ignored.

The members of the structure are:

`init`

This specifies a callback function which the application will call when it has determined this field will be used. (In the case of **skpinRegFilter()**, the function is called for all registered filters.) The application calls the function before processing data. It may be NULL; the signature of the callback function is

```
skplugin_err_t init(
    void *cbdata);
```

`cleanup`

When this callback function is not NULL, the application will call it after all records have been processed. It has the same signature as the `init` function.

`column_width`

The number of characters (not including trailing NUL) required to hold a string representation of the longest value of the field. This value can be 0 if not used (e.g., **rwsort** does not print fields), or if it will be set later using **skpinSetFieldWidths()**.

`bin_bytes`

The number of bytes (octets) required to hold a binary representation of a value of the field. This value can be 0 if not used (e.g., **rwcut** does not use binary values), or if it will be set later using **skpinSetFieldWidths()**.

`rec_to_text`

The **rwcut** application uses this callback function to fetch the textual value for the field given a SiLK Flow record. The signature of this function is

```
skplugin_err_t rec_to_text(
    const rwRec  *rec,
    char         *dest,
    size_t        width,
    void         *cbdata,
    void        **extra);
```

The callback function should fill the character array `dest` with the textual value, and the value should be NUL-terminated. `width` specifies the overall size of `dest`, and it may not have the same value as specified by the `column_width` member. For proper formatting, the callback function should write no more than `column_width` characters into `dest`. Note that if an application requires a `rec_to_bin` function and `rec_to_bin` is NULL, the application will use `rec_to_text` if it is provided. The application will use `column_width` as the width for binary values (zeroing out the destination area before it is written to).

`rec_to_bin`

This callback function is used by the application to fetch the binary value for this field given the SiLK Flow record. The signature of this function is:

```
skplugin_err_t rec_to_bin(
    const rwRec  *rec,
    uint8_t      *dest,
    void         *cbdata,
    void        **extra);
```

The callback function should write exactly `bin_bytes` of data into `dest` (where `bin_bytes` was specified in the call to **skpinRegField()** or **skpinSetFieldWidths()**). See also the `rec_to_text` member.

`add_rec_to_bin`

This callback function is used by **rwuniq** and **rwstats** when computing aggregate value fields. The application expects this function to get the binary value for this field from the SiLK Flow record and merge it (e.g., add it) to the current value. That is, the function should update the value in `current_and_new_value` with the value that comes from the current `rec`. The signature is:

```
skplugin_err_t add_rec_to_bin(
    const rwRec  *rec,
    uint8_t      *current_and_new_value,
    void         *cbdata,
    void        **extra);
```

The callback function should write exactly `bin_bytes` of data into `current_and_new_value`.

`bin_to_text`

This callback function is used to get a textual representation of a binary value that was set by a prior call to the `rec_to_bin` or `add_rec_to_bin` functions. The function signature is

```
skplugin_err_t bin_to_text(
    const uint8_t *bin,
    char          *dest,
    size_t         width,
    void          *cbdata);
```

The binary input value is in `bin`, and it is exactly `bin_bytes` in length. The textual output must be written to `dest`. The overall size of `dest` is given by `width`, which may be different than the `column_width` value that was previously specified. For proper formatting, the callback function should write no more than `column_width` characters into `dest`.

`bin_merge`

When **rwstats** and **rwuniq** are unable to store all values in memory, the applications write their current state to temporary files on disk. Once all input data has been processed, the temporary files are combined to produce the output. When a key appears in multiple temporary files, the aggregate values must be merged (for example, the byte count for two keys would be added). This callback function is used to merge aggregate value fields defined by the plug-in. The function signature is below. The `src1_and_dest` parameter will contain a binary aggregate value from one of the files, and the `src2` parameter a value from the other. These should be combined and the (binary) result written to `src1_and_dest`. The byte length of both parameters is `bin_bytes`.

```
skplugin_err_t bin_merge(
    uint8_t        *src1_and_dest,
    const uint8_t  *src2,
    void           *cbdata);
```

`bin_compare`

This callback function is used by **rwstats** when determining the top-N (or bottom-N) bins based on the binary aggregate values. The function accepts two binary values, `value_a` and `value_b`, each of length `bin_bytes`. The function must set `cmp_result` to an integer less than 0, equal 0, or greater than 0 to indicate whether `value_a` is less than, equal to, or greater than `value_b`, respectively. If this function is NULL, **memcmp()** will be used on the binary values instead.

```
skplugin_err_t bin_compare(
    int            *cmp_result,
    const uint8_t  *value_a,
    const uint8_t  *value_b,
    void           *cbdata);
```

`filter`

This callback function is only required when the plug-in will be used by **rwfilter**, as described above. When defining a field, `filter` is ignored.

`transform`

This callback function is only required when the plug-in will be used by **rwptoflow**. This callback allows the plug-in to modify the SiLK Flow record, `rec`, before it is written to the output. The callback function should modify `rec` in place; the signature is

```
skplugin_err_t transform(
    rwRec  *rec,
    void   *cbdata,
    void  **extra);
```

`initial`

When the `initial` member is not NULL, it should point to a value containing at least `bin_bytes` bytes. These bytes will be used to initialize the binary aggregate value. As an example use case, when the plug-in is computing a minimum, it may choose to initialize the field to contain the maximum value. When `initial` is NULL, binary aggregate values are initialized using **bzero()**.

`extra`

This member is usually NULL. When not NULL, it points to a NULL-terminated constant array of strings representing "extra arguments". These are not often used, and they will not be discussed in this manual page.

Once a field is registered, you may make changes to it by calling the additional functions described below. In each of these functions, the `field` parameter is the handle returned when the field was registered.

By default, the `name` will also be used as the field's title. To specify a different title, the plug-in may call

```
skplugin_err_t skpinSetFieldTitle(
    skplugin_field_t    field,
    const char          title);
```

To create an alternate name for the field (that is, a name that can be used in the **--fields** or **--values** switches) call

```
skplugin_err_t skpinAddFieldAlias(
    skplugin_field_t    field,
    const char          alias);
```

To set or modify the textual and binary widths for a field, use the following function. This function should called in the field's `init` callback function.

```
skplugin_err_t skpinSetFieldWidths(
    skplugin_field_t    field,
    size_t              field_width_text,
    size_t              field_width_bin);
```

The following table shows when a member of the `skplugin_callbacks_t` structure is required or optional. (Where the table shows `column_width` and `bin_bytes` as required, the values can be set in the structure or via the **skpinSetFieldWidths()** function.)

	rwfilter	rwcut	rwgroup	rwsort	rwstats	rwuniq	rwptoflow
init	r	f	f	f	f,a	f,a	r
cleanup	r	f	f	f	f,a	f,a	r
column_width	.	F	.	.	F,A	F,A	.
bin_bytes	.	.	F	F	F,A	F,A	.
rec_to_text	.	F	.	.	.	.	.
rec_to_bin	.	.	F	F	F	F	.
add_rec_to_bin	.	.	.	.	A	A	.
bin_to_text	.	.	.	.	F,A	F,A	.
bin_merge	.	.	.	.	A	A	.
bin_compare	.	.	.	.	A	.	.
initial	.	.	.	.	a	a	.
filter	R	.	.	.	.	.	.
transform	.	.	.	.	.	.	R
extra	r	f	f	f	f,a	f,a	r

The legend is

F

required for a key field

A

required for an aggregate value field

R

required for a non-field application (e.g., **rwfilter**)

f

optional for a key field

a

optional for an aggregate value field

r

optional for a non-field application

.

ignored

Miscellaneous functions

The following registers a cleanup function for the plug-in. This function will be called by the application after any field- or filter-specific cleanup functions are called. Specifically, this is the last callback that the application will invoke on a plug-in.

```
skplugin_err_t skpinRegCleanup(
    skplugin_cleanup_fn_t cleanup);
```

The signature of the `cleanup` function is:

```
void cleanup(void);
```

The plug-in author should invoke the following function to tell **rwfilter** that this plug-in is not thread safe. Calling this function causes **rwfilter** not use multiple threads; as such, this function should only be called when the plug-in has registered an active filter function.

```
void skpinSetThreadNonSafe(void);
```

Compiling the plug-in

Once you have finished writing the C code for the plug-in, save it in a file. The following uses the name *my-plugin.c* for the name of this file.

In the following, the leading dollar sign (`$`) followed by a space represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

When compiling a plug-in, you should use the same compiler and compiler-options as when SiLK was compiled. The **silk_config(1)** utility can be used to obtain that information. To store the compiler used to compile SiLK into the variable `sk_cc`, specify the following at a shell prompt (note that those are backquotes, and this assumes a Bourne-compatible shell):

```
$ sk_cc=‘silk_config --compiler‘
```

To get the compiler flags used to compile SiLK:

```
$ sk_cflags=‘silk_config --cflags‘
```

Using those two variables, you can now compile the plug-in. The following will work on Linux and Mac OS X:

```
$ $sk_cc $sk_cflags -shared -o my-plugin.so my-plugin.c
```

For Mac OS X:

```
$ $sk_cc $sk_cflags -bundle -flat_namespace -undefined suppress    \
       -o my-plugin.so my-plugin.c
```

If there are compilation errors, fix them and compile again.

Notes: The preceding assumed you were building the plug-in after having installed SiLK. The paths given by **silk_config** do not work if SiLK has not been installed. To compile the plug-in, you must have access to the SiLK header files. (If you are using an RPM installation of SiLK, ensure that the `silk-devel` RPM is installed.)

Once you have created the *my-plugin.so* file, you can load it into an application by using the **--plugin** switch on the application as shown in the SYNOPSIS. When loading a plug-in from the current directly, it is best to prefix the filename with `./`:

```
$ rwcut --plugin=./my-plugin.so ...
```

If there are problems loading the plug-in into the application, you can trace the actions the application is doing by setting the SILK_PLUGIN_DEBUG environment variable:

```
$ SILK_PLUGIN_DEBUG=1  rwcut --plugin=./my-plugin.so ...
```

EXAMPLES

rwfilter

Suppose you want to find traffic destined to a particular host, 10.0.0.23, that is either ICMP or coming from 1434/udp. If you attempt to use:

```
$ rwfilter --daddr=10.0.0.23 --proto=1,17 --sport=1434     \
       --pass=outfile.rw  flowrec.rw
```

the **--sport** option will not match any of the ICMP traffic, and your result will not contain ICMP records. To avoid having to use two invocations of **rwfilter**, you can create the following plug-in to do the entire check in a single pass:

```
#include <silk/silk.h>
#include <silk/rwrec.h>
#include <silk/skipaddr.h>
#include <silk/skplugin.h>
#include <silk/utils.h>
```

```
/* These variables specify the version of the SiLK plug-in API. */
#define PLUGIN_API_VERSION_MAJOR 1
#define PLUGIN_API_VERSION_MINOR 0

/* ip to search for */
static skipaddr_t ipaddr;

/*
 *  status = filter(rwrec, reg_data, extra);
 *
 *    The function should examine the SiLK flow record and return
 *    SKPLUGIN_FILTER_PASS to write the rwRec to the
 *    pass-destination(s) or SKPLUGIN_FILTER_FAIL to write it to the
 *    fail-destination(s).
 */
static skplugin_err_t filter(
    const rwRec    *rwrec,
    void           *reg_data,
    void          **extra)
{
    skipaddr_t dip;

    rwRecMemGetDIP(rwrec, &dip);
    if (0 == skipaddrCompare(&dip, &ipaddr)
        && (rwRecGetProto(rwrec) == 1
            || (rwRecGetProto(rwrec) == 17
                && rwRecGetSPort(rwrec) == 1434)))
    {
        return SKPLUGIN_FILTER_PASS;
    }
    return SKPLUGIN_FILTER_FAIL;
}

/* The set-up function that the application will call. */
skplugin_err_t SKPLUGIN_SETUP_FN(
    uint16_t    major_version,
    uint16_t    minor_version,
    void       *plug_in_data)
{
    uint32_t ipv4;
    skplugin_err_t rv;
    skplugin_callbacks_t regdata;

    /* Check the plug-in API version */
    rv = skpinSimpleCheckVersion(major_version, minor_version,
                                 PLUGIN_API_VERSION_MAJOR,
                                 PLUGIN_API_VERSION_MINOR,
                                 skAppPrintErr);
    if (rv != SKPLUGIN_OK) {
        return rv;
    }
```

```
    /* set global ipaddr */
    ipv4 =  ((10 << 24) | 23);
    skipaddrSetV4(&ipaddr, &ipv4);

    /* register the filter */
    memset(&regdata, 0, sizeof(regdata));
    regdata.filter = filter;
    return skpinRegFilter(NULL, &regdata, NULL);
}
```

Once this file is created and compiled, you can use it from **rwfilter** as shown here:

```
$ rwfilter --plugin=./my-plugin.so --pass=outfile.rw  flowrec.rw
```

Additional examples

For additional examples, see the source files in *silk-VERSION/src/plugins*.

ENVIRONMENT

SILK_PATH

This environment variable gives the root of the install tree. When searching for plug-ins, a SiLK application may use this environment variable. See the FILES section for details.

SILK_PLUGIN_DEBUG

When set to 1, the SiLK applications print status messages to the standard error as they attempt to find and open each plug-in. In addition, when an attempt to register a field fails, the application prints a message specifying the additional function(s) that must be defined to register the field in the application. Be aware that the output can be rather verbose.

FILES

${SILK_PATH}/lib64/silk/

${SILK_PATH}/lib64/

${SILK_PATH}/lib/silk/

${SILK_PATH}/lib/

/usr/local/lib64/silk/

/usr/local/lib64/

/usr/local/lib/silk/

/usr/local/lib/

Directories that a SiLK application checks when attempting to load a plug-in.

SEE ALSO

rwfilter(1), **rwcut(1)**, **rwgroup(1)**, **rwsort(1)**, **rwstats(1)**, **rwuniq(1)**, **silk_config(1)**, **rwptoflow(1)**, **pysilk(3)**, **silkpython(3)**, **flowrate(3)**, **silk(7)**, **pcap(3)**

silkpython

SiLK Python plug-in

SYNOPSIS

```
rwfilter --python-file=FILENAME [--python-file=FILENAME ...] ...

rwfilter --python-expr=PYTHON_EXPRESSION ...

rwcut --python-file=FILENAME [--python-file=FILENAME ...]
      --fields=FIELDS ...

rwgroup --python-file=FILENAME [--python-file=FILENAME ...]
      --id-fields=FIELDS ...

rwsort --python-file=FILENAME [--python-file=FILENAME ...]
      --fields=FIELDS ...

rwstats --python-file=FILENAME [--python-file=FILENAME ...]
      --fields=FIELDS --values=VALUES ...

rwuniq --python-file=FILENAME [--python-file=FILENAME ...]
      --fields=FIELDS --values=VALUES ...
```

DESCRIPTION

The SiLK Python plug-in provides a way to use PySiLK (the SiLK extension for **python(1)** described in **pysilk(3)**) to extend the capability of several SiLK tools.

- In **rwfilter(1)**, new partitioning rules can be defined in PySiLK to determine whether a SiLK Flow record is written to the **--pass-destination** or **--fail-destination**.
- In **rwcut(1)**, new fields can be defined in PySiLK and displayed for each record.
- New fields can also be defined in **rwgroup(1)** and **rwsort(1)**. These fields are used as part of the key when grouping or sorting the records.
- For **rwstats(1)** and **rwuniq(1)**, two types of fields can be defined: *Key fields* are used to categorize the SiLK Flow records into bins, and *aggregate value fields* compute a value across all the SiLK Flow records that are categorized into a bin. (An example of a built-in aggregate value field is the number of packets that were seen for all flow records that match a particular key.)

To extend the SiLK tools using PySiLK, the user writes a Python file that calls Python functions defined in the **silk.plugin** Python module and described in this manual page. When the user specifies the **--python-file** switch to a SiLK application, the application loads the Python file and makes the new functionality available.

The following sections will describe

- how to create a command line switch with PySiLK that allows one to modify the run-time behavior of their PySiLK code
- how to use PySiLK with **rwfilter**
- a simple API for creating fields in **rwcut**, **rwgroup**, **rwsort**, **rwstats**, and **rwuniq**
- the advanced API for creating fields in those applications

Typically you will not need to explicitly import the **silk.plugin** module, since the **--python-file** switch does this for you. In a module used by a Python plug-in, the module can gain access to the functions defined in this manual page by importing them from **silk.plugin**:

```
from silk.plugin import *
```

Hint: If you want to check whether the Python code in *FILENAME* is defining the switches and fields you expect, you can load the Python file and examine the output of **--help**, for example:

```
rwcut --python-file=FILENAME --help
```

User-defined command line switches

Command line switches can be added and handled from within a SiLK Python plug-in. In order to add a new switch, use the following function:

register_switch(*switch_name***, handler=***handler_func***,** [**arg=***needs_arg*], [**help=***help_string*])

switch_name

Provides the name of the switch you are registering, a string. Do not include the leading `--` in the name. If a switch already exists with the name *switch_name*, the application will exit with an error message.

handler_func

handler_func(*[string]***)**. Names a function that will be called by the application while it is processing its command line if and only if the command line includes the switch **--switch_name**. (If the switch is not given, the *handler_func* function will not be called.) When the **arg** parameter is specified and its value is **False**, the *handler_func* function will be called with no arguments. Otherwise, the *handler_func* function will be called with a single argument: a string representing the value the user passed to the **--switch_name** switch. The return value from this function is ignored. Note that the **register_switch()** function requires a **handler** argument which must be passed by keyword.

needs_arg

Specifies a boolean value that determines whether the user must specify an argument to **--switch_name**, and determines whether the *handler_func* function should expect an argument. When **arg** is not specified or *needs_arg* is **True**, the user must specify an argument to **--switch_name** and the *handler_func* function will be called with a single argument. When *needs_arg* is **False**, it is an error to specify an argument to **--switch_name** and *handler_func* will be called with no arguments.

help_string

Provides the usage text to print describing this switch when the user runs the application with the **--help** switch. This argument is optional; when it is not provided, a simple "No help for this switch" message is printed.

rwfilter usage

When used in conjunction with **rwfilter(1)**, the SiLK Python plug-in allows users to define arbitrary partitioning criteria using the SiLK extension to the Python programming language. To use this capability, the user creates a Python file and specifies its name with the **--python-file** switch in **rwfilter**. The file should call the **register_filter()** function for each filter that it wants to create:

register_filter(*filter_func***,** [**finalize=***finalize_func*], [**initialize=***initialize_func*])

filter_func

Boolean = **filter_func(***silk.RWRec***)**. Names a function that must accept a single argument, a **silk.RWRec** object (see **pysilk(3)**). When the **rwfilter** program is run, it finds the records that match the selection options, and hands each record to the built-in partitioning switches. A record that passes all of the built-in switches is handed to the first Python **filter_func()** function as an **RWRec** object. The return value of the function determines what happens to the record. The record fails the **filter_func()** function (and the record is immediately written to the **--fail-destination**, if specified) when the function returns one of the following: **False**, **None**, numeric zero of any type, an empty string, or an empty container (including strings, tuples, lists, dictionaries, sets, and frozensets). If the function returns any other value, the record passes the first **filter_func()** function, and the record is handed to the next Python **filter_func()** function. If all **filter_func()** functions pass the record, the record is written to the **--pass-destination**, if specified. (Note that when the **--plugin** switch is present, the code it specifies will be called after the PySiLK code.)

initialize_func

initialize_func(). Names a function that takes no arguments. When this function is specified, is will be called after **rwfilter** has completed its argument processing, and just before **rwfilter** opens the first input file. The return value of this function is ignored.

finalize_func

finalize_func(). Names a function that takes no arguments. When this function is specified, it will be called after all flow records have been processed. One use of the these functions is to print any statistics that the **filter_func()** function was computing. The return value from this function is ignored.

If **register_filter()** is called multiple times, the **filter_func()**, **initialize_func()**, and **finalize_func()** functions will be invoked in the order in which the **register_filter()** functions were seen.

NOTE: For backwards compatibility, when the file named by **--python-file** does not call **register_filter()**, **rwfilter** will search the Python file for functions named **rwfilter()** and **finalize()**. If it finds the **rwfilter()** function, **rwfilter** will act as if the file contained:

```
register_filter(rwfilter, finalize=finalize)
```

The **--python-file** switch requires the user to create a file containing Python code. To allow the user to write a small filtering check in Python, **rwfilter** supports the **--python-expr** switch. The value of the switch should be a Python expression whose result determines whether a given record passes or fails, using the same criterion as the **filter_func()** function described above. In the expression, the variable `rec` is bound to the current **silk.RWRec** object. There is no support for the **initialize_func()** and **finalize_func()** functions. The user may consider **--python-expr**=*PYTHON_EXPRESSION* as being implemented by

```
from silk import *
def temp_filter(rec):
    return (PYTHON_EXPRESSION)
```

```
register_filter(temp_filter)
```

The **--python-file** and **--python-expr** switches allow for much flexibility but at the cost of speed: converting a SiLK Flow record into an **RWRec** is expensive relative to most operations in **rwfilter**. The user should use **rwfilter**'s built-in partitioning switches to whittle down the input as much as possible, and only use the Python code to do what is difficult or impossible to do otherwise.

Simple field registration functions

The **silk.plugin** module defines a function that can be used to define fields for use in **rwcut**, **rwgroup**, **rwsort**, **rwstats**, and **rwuniq**. That function is powerful, but it is also complex. To make it easy to define fields for the common cases, the **silk.plugin** provides the functions described in this section that create a key field or an aggregate value field. The advanced function is described later in this manual page (Advanced field registration function).

Once you have created a key field or aggregate value field, you must include the field's name in the argument to the **--fields** or **--values** switch to tell the application to use the field.

Integer key field

The following function is used to create a key field whose value is an unsigned integer.

register_int_field(*field_name***,** *int_function***,** *min***,** *max***,** [*width*]**)**

field_name

The name of the new field, a string. If you attempt to add a key field that already exists, you will get an an error message.

int_function

int **= int_function(***silk.RWRec***)**. A function that accepts a **silk.RWRec** object as its sole argument, and returns an unsigned integer which represents the value of this field for the given record.

min

A number representing the minimum integer value for the field. If *int_function* returns a value less than *min*, an error is raised.

max

A number representing the maximum integer value for the field. If *int_function* returns a value greater than *max*, an error is raised.

width

The column width to use when displaying the field. This parameter is optional; the default is the number of digits necessary to display the integer *max*.

IPv4 address key field

This function is used to create a key field whose value is an IPv4 address. (See also **register_ip_field()**).

register_ipv4_field(*field_name***,** *ipv4_function***,** [*width*]**)**

field_name

The name of the new field, a string. If you attempt to add a key field that already exists, you will get an an error message.

ipv4_function

silk.IPv4Addr = **ipv4_function(***silk.RWRec***)**. A function that accepts a **silk.RWRec** object as its sole argument, and returns a **silk.IPv4Addr** object. This **IPv4Addr** object will be the IPv4 address that represents the value of this field for the given record.

width

The column width to use when displaying the field. This parameter is optional, and it defaults to 15.

IP address key field

The next function is used to create a key field whose value is an IPv4 or IPv6 address.

register_ip_field(*field_name***,** *ip_function***,** [*width*])

field_name

The name of the new field, a string. If you attempt to add a key field that already exists, you will get an an error message.

ip_function

silk.IPAddr = **ip_function(***silk.RWRec***)**. A function that accepts a **silk.RWRec** object as its sole argument, and returns a **silk.IPAddr** object which represents the value of this field for the given record.

width

The column width to use when displaying the field. This parameter is optional. The default width is 39.

This key field requires more memory internally than fields registered by the **register_ipv4_field()** function. If SiLK is compiled without IPv6 support, **register_ip_field()** works exactly like **register_ipv4_field()**, including the default width of 15.

Enumerated object key field

The following function is used to create a key field whose value is any Python object. The maximum number of different objects that can be represented is 4,294,967,296, or 2^32.

register_enum_field(*field_name***,** *enum_function***,** *width***,** [*ordering*])

field_name

The name of the new field, a string. If you attempt to add a key field that already exists, you will get an an error message.

enum_function

object = **enum_function(***silk.RWRec***)**. A function that accepts a **silk.RWRec** object as its sole argument, and returns a Python object which represents the value of this field for the given record. For typical usage, the Python objects returned by the *enum_function* will be strings representing some categorical value.

width

The column width to use when displaying this field. The parameter is required.

ordering

A list of objects used to determine ordering for **rwsort** and **rwuniq**. This parameter is optional. If specified, it lists the objects in the order in which they should be sorted. If the *enum_function* returns a object that is not in *ordering*, the object will be sorted after all the objects in *ordering*.

Integer sum aggregate value field

This function is used to create an aggregate value field that maintains a running unsigned integer sum.

register_int_sum_aggregator(*agg_value_name***,** *int_function***,** [*max_sum*]**,** [*width*]**)**

agg_value_name

The name of the new aggregate value field, a string. The *agg_value_name* must be unique among all aggregate values, but an aggregate value field and key field can have the same name.

int_function

int **= int_function(***silk.RWRec***)**. A function that accepts a **silk.RWRec** object as its sole argument, and returns an unsigned integer which represents the value that should be added to the running sum for the current bin.

max_sum

The maximum possible sum. This parameter is optional; if not specified, the default is 2^64-1 (18,446,744,073,709,551,615).

width

The column width to use when displaying the aggregate value. This parameter is optional. The default is the number of digits necessary to display *max_sum*.

Integer maximum aggregate value field

The following function is used to create an aggregate value field that maintains the maximum unsigned integer value.

register_int_max_aggregator(*agg_value_name***,** *int_function***,** [*max_max*]**,** [*width*]**)**

agg_value_name

The name of the new aggregate value field, a string. The *agg_value_name* must be unique among all aggregate values, but an aggregate value field and key field can have the same name.

int_function

int **= int_function(***silk.RWRec***)**. A function that accepts a **silk.RWRec** object as its sole argument, and returns an integer which represents the value that should be considered for the current highest value for the current bin.

max_max

The maximum possible value for the maximum. This parameter is optional; if not specified, the default is 2^64-1 (18,446,744,073,709,551,615).

width

The column width to use when displaying the aggregate value. This parameter is optional. The default is the number of digits necessary to display *max_max*.

Integer minimum aggregate value field

This function is used to create an aggregate value field that maintains the minimum unsigned integer value.

register_int_min_aggregator(*agg_value_name***,** *int_function***,** [*max_min*]**,** [*width*]**)**

agg_value_name

The name of the new aggregate value field, a string. The *agg_value_name* must be unique among all aggregate values, but an aggregate value field and key field can have the same name.

int_function

int **= int_function(***silk.RWRec***)**. A function that accepts a **silk.RWRec** object as its sole argument, and returns an integer which represents the value that should be considered for the current lowest value for the current bin.

max_min

The maximum possible value for the minimum. When this optional parameter is not specified, the default is 2^64-1 (18,446,744,073,709,551,615).

width

The column width to use when displaying the aggregate value. This parameter is optional. The default is the number of digits necessary to display *max_min*.

Advanced field registration function

The previous section provided functions to register a key field or an aggregate value field when dealing with common objects. When you need to use a complex object, or you want more control over how the object is handled in PySiLK, you can use the **register_field()** function described in this section.

Many of the arguments to the **register_field()** function are callback functions that you must create and that the application will invoke. (The simple registration functions above have already taken care of defining these callback functions.)

Often the callback functions for handling fields will either take (as a parameter) or return a representation of a numeric value that can be processed from C. The most efficient way to handle these representations is as a string containing binary characters, including the null byte. We will use the term "byte sequence" for these representations; other possible terms include "array of bytes", "byte strings", or "binary values". For hints on creating byte sequences from Python, see the Byte sequences section below.

To define a new field or aggregate value, the user calls:

register_field(*field_name***,** [**add_rec_to_bin=***add_rec_to_bin_func***,**] [**bin_compare=***bin_compare_func***,**] [**bin_bytes=***bin_bytes_value***,**] [**bin_merge=***bin_merge_func***,**] [**bin_to_text=***bin_to_text_func***,**] [**column_width=***column_width_value***,**] [**description=***description_string***,**] [**initial_value=***initial_value***,**] [**initialize=***initialize_func***,**] [**rec_to_bin=***rec_to_bin_func***,**] [**rec_to_text=***rec_to_text_func*]**)**

Although the keyword arguments to **register_field()** are all optional from Python's perspective, certain keyword arguments must be present before an application will define the key or aggregate value. The following table summarizes the keyword arguments used by each application. An `F` means the argument is required for a key field, an `A` means the argument is required for an aggregate value field, `f` and `a` mean the application will use the argument for a key field or an aggregate value if the argument is present, and a dot means the application completely ignores the argument.

	rwcut	rwgroup	rwsort	rwstats	rwuniq
add_rec_to_bin	.	.	.	A	A
bin_compare	.	.	.	A	.
bin_bytes	.	F	F	F,A	F,A
bin_merge	.	.	.	A	A
bin_to_text	.	.	.	F,A	F,A
column_width	F	.	.	F,A	F,A
description	f	f	f	f,a	f,a
initial_value	.	.	.	a	a
initialize	f	f	f	f,a	f,a
rec_to_bin	.	F	F	F	F
rec_to_text	F	.	.	.	.

The following sections describe how to use **register_field()** in each application.

rwcut usage

The purpose of **rwcut(1)** is to print attributes of (or attributes derived from) every SiLK record it reads as input. A plug-in used by **rwcut** must produce a printable (textual) attribute from a SiLK record. To define a new attribute, the **register_field()** method should be called as shown:

register_field(*field_name***, column_width=***column_width_value***, rec_to_text=***rec_to_text_func***,** [**description=***description_string***,**] [**initialize=***initialize_func*])

field_name

Names the field being defined, a string. If you attempt to add a field that already exists, you will get an an error message. To display the field, include *field_name* in the argument to the **--fields** switch.

column_width_value

Specifies the length of the longest printable representation. **rwcut** will use it as the width for the *field_name* column when columnar output is selected.

rec_to_text_func

string = **rec_to_text_func(***silk.RWRec***)**. Names a callback function that takes a **silk.RWRec** object as its sole argument and produces a printable representation of the field being defined. The length of the returned text should not be greater than *column_width_value*. If the value returned from this function is not a string, the returned value is converted to a string by the Python **str()** function.

description_string

Provides a string giving a brief description of the field, suitable for printing in **--help-fields** output. This argument is optional.

initialize_func

initialize_func(). Names a callback function that will be invoked after the application has completed its argument processing, and just before it opens the first input file. This function is only called when **--fields** includes *field_name*. The function takes no arguments and its return value is ignored. This argument is optional.

If the **rec_to_text** argument is not present, the **register_field()** function will do nothing when called from **rwcut**. If the **column_width** argument is missing, **rwcut** will complain that the textual width of the plug-in field is 0.

rwgroup and rwsort usage

The **rwsort(1)** tool sorts SiLK records by their attributes or attributes derived from them. **rwgroup(1)** reads sorted SiLK records and writes a common value into the next hop IP field of all records that have common attributes. The output from both of these tools is a stream of SiLK records (the output typically includes every record that was read as input). A plug-in used by these tools must return a value that the application can use internally to compare records. To define a new field that may be included in the **--id-fields** switch to **rwgroup** or the **--fields** switch to **rwsort**, the **register_field()** method should be invoked as follows:

register_field(*field_name***, bin_bytes=***bin_bytes_value***, rec_to_bin=***rec_to_bin_func***,** [**description=***description_string***,**] [**initialize=***initialize_func*])

field_name

Names the field being defined, a string. If you attempt to add a field that already exists, you will get an an error message. To have **rwgroup** or **rwsort** use this field, include *field_name* in the argument to **--id-fields** or **--fields**.

bin_bytes_value

Specifies a positive integer giving the length, in bytes, of the byte sequence that the **rec_to_bin_func()** function produces; the byte sequence must be exactly this length.

rec_to_bin_func

byte-sequence = **rec_to_bin_func(***silk.RWRec***)**. Names a callback function that takes a **silk.RWRec** object and returns a byte sequence that represents the field being defined. The returned value should be exactly *bin_bytes_value* bytes long. For proper grouping or sorting, the byte sequence should be returned in network byte order (i.e., big endian).

description_string

Provides a string giving a brief description of the field, suitable for printing in **--help-fields** output. This argument is optional.

initialize_func

initialize_func(). Names a callback function that will be invoked after the application has completed its argument processing, and just before it opens the first input file. This function is only called when *field_name* is included in the list of fields. The function takes no arguments and its return value is ignored. This argument is optional.

If the **rec_to_bin** argument is not present, the **register_field()** function will do nothing when called from **rwgroup** or **rwsort**. If the **bin_bytes** argument is missing, **rwgroup** or **rwsort** will complain that the binary width of the plug-in field is 0.

rwstats and rwuniq usage

rwstats(1) and **rwuniq(1)** group SiLK records into bins based on key fields. Once a record is matched to a bin, the record is used to update the *aggregate values* (e.g., the sum of bytes) that are being computed, and the record is discarded. Once all records have been processed, the key fields and the aggregate values are printed.

Key Field

A plug-in used by **rwstats** or **rwuniq** for creating a new key field must return a value that the application can use internally to compare records, and there must be a function that converts that value to a printable representation. The following invocation of **register_field()** will produce a key field that can be used in the **--fields** switch of **rwstats** or **rwuniq**:

register_field(*field_name***, bin_bytes=***bin_bytes_value***, bin_to_text=***bin_to_text_func***, column_width=***column_width_value***, rec_to_bin=***rec_to_bin_func***,** [**description=***description_string***,**] [**initialize=***initialize_func*])

The arguments are:

field_name

Contains the name of the field being defined, a string. If you attempt to add a field that already exists, you will get an an error message. The field will only be active when *field_name* is specified as an argument to **--fields**.

bin_bytes_value

Contains a positive integer giving the length, in bytes, of the byte sequence that the **rec_to_bin_func()** function produces and that the **bin_to_text_func()** function accepts. The byte sequences must be exactly this length.

bin_to_text_func

string = **bin_to_text_func(***byte-sequence***)**. Names a callback function that takes a byte sequence, of length *bin_bytes_value*, as produced by the **rec_to_bin_func()** function and returns a printable representation of the byte sequence. The length of the text should be no longer than the value specified by **column_width**. If the value returned from this function is not a string, the returned value is converted to a string by the Python **str()** function.

column_width_value

Contains a positive integer specifying the length of the longest textual field that the **bin_to_text_func()** callback function returns. This length will used as the column width when columnar output is requested.

rec_to_bin_func

byte-sequence = **rec_to_bin_func(***silk.RWRec***)**. Names a callback function that takes a **silk.RWRec** object and returns a byte sequence that represents the field being defined. The returned value should be exactly *bin_bytes_value* bytes long. For proper sorting, the byte sequence should be returned in network byte order (i.e., big endian).

description_string

Provides a string giving a brief description of the field, suitable for printing in **--help-fields** output. This argument is optional.

initialize_func

initialize_func(). Names a callback function that is called after the command line arguments have been processed, and before opening the first file. This function is only called when **--fields** includes *field_name*. The function takes no arguments and its return value is ignored. This argument is optional.

Aggregate Value

A plug-in used by **rwstats** or **rwuniq** for creating a new aggregate value must be able to use a SiLK record to update an aggregate value, take two aggregate values and merge them to a new value, and convert that aggregate value to a printable representation. To use an aggregate value for ordering the bins in **rwstats**, the

plug-in must also define a function to compare two aggregate values. The aggregate values are represented as byte sequences.

To define a new aggregate value in **rwstats**, the user calls:

register_field(*agg_value_name***, add_rec_to_bin=***add_rec_to_bin_func***, bin_bytes=***bin_bytes_value***, bin_merge=***bin_merge_func***, bin_to_text=***bin_to_text_func***, column_width=***column_width_value***,** [**bin_compare=***bin_compare_func***,**] [**description=***description_string***,**] [**initial_value=***initial_value***,**] [**initialize=***initialize_func*])

The call to define a new aggregate value in **rwuniq** is nearly identical:

register_field(*agg_value_name***, add_rec_to_bin=***add_rec_to_bin_func***, bin_bytes=***bin_bytes_value***, bin_merge=***bin_merge_func***, bin_to_text=***bin_to_text_func***, column_width=***column_width_value***,** [**description=***description_string***,**] [**initial_value=***initial_value***,**] [**initialize=***initialize_func*])

The arguments are:

agg_value_name

Contains the name of the aggregate value field being defined, a string. The name of value must be unique among all aggregate values, but an aggregate value field and key field can have the same name. The value will only be active when *agg_value_name* is specified as an argument to **--values**.

add_rec_to_bin_func

byte-sequence = **add_rec_to_bin_func(***silk.RWRec***,** *byte-sequence***)**. Names a callback function whose two arguments are a **silk.RWRec** object and an aggregate value. The function updates the aggregate value with data from the record and returns a new aggregate value. Both aggregate values are represented as byte sequences of exactly *bin_bytes_value* bytes.

bin_bytes_value

Contains a positive integer representing the length, in bytes, of the binary aggregate value used by the various callback functions. Every byte sequence for this field must be exactly this length, and it also governs the length of the byte sequence specified by **initial_value**.

bin_merge_func

byte-sequence = **bin_merge_func(***byte-sequence***,** *byte-sequence***)**. Names a callback function which returns the result of merging two binary aggregate values into a new binary aggregate value. This merge function will often be addition; however, if the aggregate value is a bitmap, the result of merge function could be the union of the bitmaps. The function should take two byte sequence arguments and return a byte sequence, where all byte sequences are exactly *bin_bytes_value* bytes in length. If merging the aggregate values is not possible, the function should throw an exception. This function is used when the data structure used by **rwstats** or **rwuniq** runs out memory. When that happens, the application writes its current state to a temporary file, empties its buffers, and continues reading records. Once all records have been processed, the application needs to merge the temporary files to produce the final output. The **bin_merge_func()** function is used when merging these binary aggregate values.

bin_to_text_func

string = **bin_to_text_func(***byte-sequence***)**. Names a callback function that takes a byte sequence representing an aggregate value as an argument and returns a printable representation of that aggregate value. The byte sequence input to **bin_to_text_func()** will be exactly *bin_bytes_value* bytes long. The length of the text should be no longer than the value specified by **column_width**. If the value returned from this function is not a string, the returned value is converted to a string by the Python **str()** function.

column_width_value

Contains a positive integer specifying the length of the longest textual field that the **bin_to_text_func()** callback function returns. This length will used as the column width when columnar output is requested.

bin_compare_func

int = **bin_compare_func(***byte-sequence***,** *byte-sequence***)**. Names a callback function that is called with two aggregate values, each represented as a byte sequence of exactly *bin_bytes_value* bytes. The function returns (1) an integer less than 0 if the first argument is less than the second, (2) an integer greater than 0 if the first is greater than the second, or (3) 0 if the two values are equal. This function is used by **rwstats** to sort the bins into top-N order.

description_string

Provides a string giving a brief description of the aggregate value, suitable for printing in **--help-fields** output. This argument is optional.

initial_value

Specifies a byte sequence representing the initial state of the binary aggregate value. This byte sequence must be of length *bin_bytes_value* bytes. If this argument is not specified, the aggregate value is set to a byte sequence containing *bin_bytes_value* null bytes.

initialize_func

initialize_func(). Names a callback function that is called after the command line arguments have been processed, and before opening the first file. This function is only called when **--values** includes *agg_value_name*. The function takes no arguments and its return value is ignored. This argument is optional.

Byte sequences

The **rwgroup**, **rwsort**, **rwstats**, and **rwuniq** programs make extensive use of "byte sequences" (a.k.a., "array of bytes", "byte strings", or "binary values") in their plug-in functions. The byte sequences are used in both key fields and aggregate values.

When used as key fields, the values can represent uniqueness or indicate sort order. Two records with the same byte sequence for a field will be considered identical with respect to that field. When sorting, the byte sequences are compared in network byte order. That is, the most significant byte is compared first, followed by the next-most-significant byte, etc. This equates to string comparison starting with the left-hand side of the string.

When used as an aggregate field, the byte sequences are expected to behave more like numbers, with the ability to take binary record and add a value to it, or to merge (e.g., add) two byte sequences outside the context of a SiLK record.

Every byte sequence has an associated length, which is passed into the **register_field()** function in the **bin_bytes** argument. The length determines how many values the byte sequence can represent. A byte sequence with a length of 1 can represent up to 256 unique values (from 0 to 255 inclusive). A byte sequence with a length of 2 can represent up to 65536 unique values (0 to 65535). To generalize, a byte sequence with a length of n can represent up to $2^(8n)$ unique values (0 to $2^(8n)$-1).

How byte sequences are represented in Python depends on the version of Python. Python represents a sequence of characters using either the **bytes** type (introduced in 2.6) or the **unicode** type. The **bytes** type can encode byte sequences while the **unicode** type cannot. In Python 2, the **str** (string) type was an alias

for **bytes**, so that any Python 2 string is in effect a byte sequence. In Python 3, **str** is an alias for **unicode**, thus Python 3 strings are unicode objects and cannot represent byte sequences.

Python does not make conversions between integers and byte sequences particularly natural. As a result, here are some pointers on how to do these conversions:

Use the bytes() and ord() methods

If you converting a single integer value that is less than 256, the easiest way to convert it to a byte sequence is to use the **bytes()** function; to convert it back, use the **ord()** function.

```
seq = bytes([num])
num = ord(seq)
```

The **bytes()** function takes a list of integers between 0 and 255 inclusive, and returns a bytes sequence of the length of that list. To convert a single byte, use a list of a single element. The **ord()** function takes a byte sequence of a single byte and returns an integer between 0 and 255.

Note: In versions of Python earlier than 2.6, use the **chr()** function instead of the **bytes()** function. It takes a single number as its argument. **chr()** will work in Python 2.6 and 2.7 as well, but there are compatibility problems in Python 3.x.

Use the struct module

When the value you are converting to a byte sequence is 255 or greater, you have to go with another option. One of the simpler options is to use Python's built-in **struct** module. With this module, you can encode a number or a set of numbers into a byte sequence and convert the result back using a **struct.Struct** object. Encoding the numbers to a byte sequence uses the object's **pack()** method. To convert that byte sequence back to the number or set of numbers, use the object's **unpack()** method. The length of the resulting byte sequences can be found in the **size** attribute of the **struct.Struct()** object. A formatting string is used to indicate how the numbers are encoded into binary. For example:

```
import struct

# Set up the format for two 64-bit numbers
two64 = struct.Struct("!QQ)
# Encode two 64-bit numbers as a byte sequence
seq = two64.pack(num1, num2)
#Unpack a byte sequence back into two 64-bit numbers
(num1, num2) = two64.unpack(seq)
#Length of the encoded byte sequence
bin_bytes = two64.size
```

In the above, `Q` represents a single unsigned 64-bit number (an unsigned long long or quad). The `!` at the beginning of the string forces network byte order. (For sort comparison purposes, always pack in network byte order.)

Here is another example, which encodes a signed 16-bit integer and a floating point number:

```
import struct

# Set up the format for a 16-bit signed integer and a float
obj = struct.Struct("!hf")
```

```
#Encode a 16-bit signed integer and a float as a byte sequence
seq = obj.pack(intval, floatval)
#Unpack a byte sequence back into a 16-bit signed integer and a float
(intval, floatval) = obj.unpack(seq)
#Length of the encoded byte sequence
bin_bytes = obj.size
```

Note that **unpack()** returns a sequence. When unpacking a single value, assign the result of **unpack** to (*variable_name*,), as shown:

```
import struct

u32 = struct.Struct("!I")
#Encode an unsigned 32-bit integer as a byte sequence
seq = u32.pack(num1)
#Unpack a byte sequence back into a unsigned 32-bit integer
(num1,) = struct.unpack(seq)
#Length of the encoded byte sequence
bin_bytes = u32.size
```

The full list of codes can be found in the Python library documentation for the **struct** module, http://docs.python.org/library/struct.html.

Note: Python versions prior to 2.5 do not include support for the **struct.Struct** object. For older versions of Python, you have to use **struct**'s functional interface. For example:

```
import struct

#Encode a 16-bit signed integer and a float as a byte sequence
seq = struct.pack("!hf", intval, floatval)
#Unpack a byte sequence back into a 16-bit signed integer and a float
(intval, floatval) = struct.unpack("!hf", seq)
#Length of the encoded byte sequence
bin_bytes = struct.calcsize("!hf")
```

This method works in Python 2.5 and above as well, but is inherently slower, as it requires re-evaluation of the format string for each packing and unpacking operation. Only use this if there is a need to inter-operate with older versions of Python.

Use the array module

The Python **array** module provides another way to create byte sequences. Beware that the **array** module does not provide an automatic way to encode the values in network byte order.

OPTIONS

The following options are available when the SiLK Python plug-in is used from **rwfilter**.

--python-file=*FILENAME*

Load the Python file *FILENAME*. The Python code may call **register_filter()** multiple times to define new partitioning functions that takes a **silk.RWRec** object as an argument. The return value

of the function determines whether the record passes the filter. For backwards compatibility, if **register_filter()** is not called and a function named **rwfilter()** exists, that function is automatically registered as the filtering function. Multiple **--python-file** switches may be used to load multiple plug-ins.

--python-expr=*PYTHON_EXPRESSION*

Pass the SiLK Flow record if the result of the processing the record with the specified *PYTHON_EXPRESSION* is true. The expression is evaluated in the following context:

- The record is represented by the variable named **rec**, which is a **silk.RWRec** object.
- There is an implicit **from silk import *** in effect.

The following options are available when the SiLK Python plug-in is used from **rwcut**, **rwgroup**, **rwsort**, **rwstats**, or **rwuniq**:

--python-file=*FILENAME*

Load the Python file *FILENAME*. The Python code may call **register_field()** multiple times to define new fields for use by the application. When used with **rwstats** or **rwuniq**, the Python code may call **register_field()** multiple times to create new aggregate fields. Multiple **--python-file** switches may be used to load multiple plug-ins.

EXAMPLES

In the following examples, the dollar sign (`$`) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

rwfilter --python-expr

Suppose you want to find traffic destined to a particular host, 10.0.0.23, that is either ICMP or coming from 1434/udp. If you attempt to use:

```
$ rwfilter --daddr=10.0.0.23 --proto=1,17 --sport=1434         \
       --pass=outfile.rw  flowrec.rw
```

the **--sport** option will not match any of the ICMP traffic, and your result will not contain ICMP records. To avoid having to use two invocations of **rwfilter**, you can use the SiLK Python plugin to do the check in a single pass:

```
$ rwfilter --daddr=10.0.0.23 --proto=1,17                      \
       --python-expr 'rec.protocol==1 or rec.sport==1434'      \
       --pass=outfile.rw  flowrec.rw
```

Since the Python code is slower than the C code used internally by **rwfilter**, we want to limit the number of records processed in Python as much as possible. We use the **rwfilter** switches to do the address check and protocol check, and in Python we only need to check whether the record is ICMP or if the source port is 1434 (if the record is not ICMP we know it is UDP because of the **--proto** switch).

rwfilter --python-file

To see all records whose protocol is different from the preceding record, use the following Python code. The code also prints a message to the standard output on completion.

```
import sys

def filter(rec):
    global lastproto
    if rec.protocol != lastproto:
        lastproto = rec.protocol
        return True
    return False

def initialize():
    global lastproto
    lastproto = None

def finalize():
    sys.stdout.write("Finished processing records.\n")

register_filter(filter, initialize = initialize, finalize = finalize)
```

The preceding file, if called *lastproto.py*, can be used like this:

```
$ rwfilter --python-file lastproto.py --pass=outfile.rw flowrec.rw
```

Note: Be careful when using a Python plug-in to write to the standard output, since the Python output could get intermingled with the output from **--pass=stdout** and corrupt the SiLK output file. In general, printing to the standard error is safer.

Command line switch

The following code registers the command line switch `count-protocols`. This switch is similar to the standard **--protocol** switch on **rwfilter**, in that it passes records whose protocol matches a value specified in a list. In addition, when **rwfilter** exits, the plug-in prints a count of the number of records that matched each specified protocol.

```
import sys
from silk.plugin import *

pro_count = {}

def proto_count(rec):
    global pro_count
    if rec.protocol in pro_count.keys():
        pro_count[rec.protocol] += 1
        return True
    return False
```

```
def print_counts():
    for p,c in pro_count.iteritems():
        sys.stderr.write("%3d|%10d|\n" % (p, c))

def parse_protocols(protocols):
    global pro_count
    for p in protocols.split(","):
        pro_count[int(p)] = 0
    register_filter(proto_count, finalize = print_counts)

register_switch("count-protocols", handler=parse_protocols,
                help="Like --proto, but prints count of flow records")
```

When this code is saved to the file *count-proto.py*, it can be used with **rwfilter** as shown to get a count of TCP and UDP flow records:

```
$ rwfilter --start-date=2008/08/08 --type=out                  \
       --python-file=count-proto.py --count-proto=6,17         \
       --print-statistics=/dev/null
```

rwfilter does not know that the plug-in will be generating output, and **rwfilter** will complain unless an output switch is given, such as **--pass** or **--print-statistics**. Since our plug-in is printing the data we want, we send the output to */dev/null*.

Create integer key field with simple API

This example creates a field that contains the sum of the source and destination port. While this value may not be interesting to display in **rwcut**, it provides a way to sort fields so traffic between two low ports will usually be sorted before traffic between a low port and a high port.

```
def port_sum(rec):
    return rec.sport + rec.dport

register_int_field("port-sum", port_sum)
```

If the above code is saved in a file named *portsum.py*, it can be used to sort traffic prior to printing it (low-port to low-port will appear first):

```
$ rwfilter --start-date=2008/08/08 --type=out,outweb       \
       --proto=6,17 --pass=stdout                          \
  | rwsort --python-file=portsum.py --fields=port-sum      \
  | rwcut
```

To see high-port to high-port traffic first, reverse the sort:

```
$ rwfilter --start-date=2008/08/08 --type=out,outweb       \
       --proto=6,17 --pass=stdout                          \
  | rwsort --python-file=portsum.py --fields=port-sum      \
       --reverse                                           \
  | rwcut
```

Create IP key field with simple API

SiLK stores uni-directional flows. For network conversations that cross the network border, the source and destination hosts are swapped depending on the direction of the flow. For analysis, you often want to know the internal and external hosts.

The following Python plug-in file defines two new fields: `internal-ip` will display the destination IP for an incoming flow, and the source IP for an outgoing flow, and `external-ip` field shows the reverse.

```
import silk

# for convenience, create lists of the types
in_types = ['in', 'inweb', 'innull', 'inicmp']
out_types = ['out', 'outweb', 'outnull', 'outicmp']

def internal(rec):
    "Returns the IP Address of the internal side of the connection"
    if rec.typename in out_types:
        return rec.sip
    else:
        return rec.dip

def external(rec):
    "Returns the IP Address of the external side of the connection"
    if rec.typename in in_types:
        return rec.sip
    else:
        return rec.dip

register_ip_field("internal-ip", internal)
register_ip_field("external-ip", external)
```

If the above code is saved in a file named *direction.py*, it can be used to show the internal and external IP addresses and flow direction for all traffic on 1434/udp from Aug 8, 2008.

```
$ rwfilter --start-date=2008/08/08 --type=all              \
       --proto=17 --aport=1434 --pass=stdout               \
  | rwcut --python-file direction.py                       \
       --fields internal-ip,external-ip,3-12
```

Create enumerated key field with simple API

This example expands the previous example. Suppose instead of printing the internal and external IP address, you wanted to group by the label associated with the internal and external addresses in a prefix map file. The **pmapfilter(3)** manual page specifies how to print labels for source and destination IP addresses, but it does not support internal and external IPs.

Here we take the previous example, add a command line switch to specify the path to a prefix map file, and have the internal and external functions return the label.

```
import silk
```

```
# for convenience, create lists of the types
in_types = ['in', 'inweb', 'innull', 'inicmp']
out_types = ['out', 'outweb', 'outnull', 'outicmp']

# handler for the --int-ext-pmap command line switch
def set_pmap(arg):
    global pmap
    pmap = silk.PrefixMap(arg)
    labels = pmap.values()
    width = max(len(x) for x in labels)
    register_enum_field("internal-label", internal, width, labels)
    register_enum_field("external-label", external, width, labels)

def internal(rec):
    "Returns the label for the internal side of the connection"
    global pmap
    if rec.typename in out_types:
        return pmap[rec.sip]
    else:
        return pmap[rec.dip]

def external(rec):
    "Returns the label for the external side of the connection"
    global pmap
    if rec.typename in in_types:
        return pmap[rec.sip]
    else:
        return pmap[rec.dip]

register_switch("int-ext-pmap", handler=set_pmap,
                help="Prefix map file for internal-label, external-label")
```

Assuming the above is saved in the file *int-ext-pmap.py*, the following will group the flows by the internal and external labels contained in the file *ip-map.pmap*.

```
$ rwfilter --start-date=2008/08/08 --type=all              \
       --proto=17 --aport=1434 --pass=stdout               \
  | rwuniq --python-file int-ext-pmap.py                   \
       --int-ext-pmap ip-map.pmap                          \
       --fields internal-label,external-label
```

Create minimum/maximum integer value field with simple API

The following example will create new aggregate fields to print the minimum and maximum byte values:

```
register_int_min_aggregator("min-bytes", lambda rec: rec.bytes,
                            (1 << 32) - 1)
register_int_max_aggregator("max-bytes", lambda rec: rec.bytes,
                            (1 << 32) - 1)
```

The **lambda** expression allows one to create an anonymous function. In this code, we need to return the number of bytes for the given record, and we can easily do that with the anonymous function. Since the SiLK bytes field is 32 bits, the maximum 32-bit number is passed the registration functions.

Assuming the code is stored in a file *bytes.py*, it can be used with **rwuniq** to see the minimum and maximum byte counts for each source IP address:

```
$ rwuniq --python-file=bytes.py --fields=sip                \
       --values=records,bytes,min-bytes,max-bytes
```

Create IP key for rwcut with advanced API

This example is similar to the simple IP example above, but it uses the advanced API. It also creates another field to indicate the direction of the flow, and it does not print the IPs when the traffic does not cross the border. Note that this code has to determine the column width itself.

```
import silk, os

# for convenience, create lists of the types
in_types = ['in', 'inweb', 'innull', 'inicmp']
out_types = ['out', 'outweb', 'outnull', 'outicmp']
internal_only = ['int2int']
external_only = ['ext2ext']

# determine the width of the IP field depending on whether SiLK
# was compiled with IPv6 support, and allow the IP_WIDTH environment
# variable to override that width.
ip_len = 15
if silk.ipv6_enabled():
    ip_len = 39
ip_len = int(os.getenv("IP_WIDTH", ip_len))

def cut_internal(rec):
    "Returns the IP Address of the internal side of the connection"
    if rec.typename in in_types:
        return rec.dip
    if rec.typename in out_types:
        return rec.sip
    if rec.typename in internal_only:
        return "both"
    if rec.typename in external_only:
        return "neither"
    return "unknown"

def cut_external(rec):
    "Returns the IP Address of the external side of the connection"
    if rec.typename in in_types:
        return rec.sip
    if rec.typename in out_types:
        return rec.dip
```

```
    if rec.typename in internal_only:
        return "neither"
    if rec.typename in external_only:
        return "both"
    return "unknown"

def internal_external_direction(rec):
    """Generates a string pointing from the sip to the dip, assuming
    internal is on the left, and external is on the right."""
    if rec.typename in in_types:
        return "<---"
    if rec.typename in out_types:
        return "--->"
    if rec.typename in internal_only:
        return "-><-"
    if rec.typename in external_only:
        return "<-->"
    return "????"

register_field("internal-ip", column_width = ip_len,
               rec_to_text = cut_internal)
register_field("external-ip", column_width = ip_len,
               rec_to_text = cut_external)
register_field("int_to_ext", column_width = 4,
               rec_to_text = internal_external_direction)
```

The **cut_internal()** and **cut_external()** functions may return an **IPAddr** object instead of a string. For those cases, the Python **str()** function is invoked automatically to convert the **IPAddr** to a string.

If the above code is saved in a file named *direction.py*, it can be used to show the internal and external IP addresses and flow direction for all traffic on 1434/udp from Aug 8, 2008.

```
$ rwfilter --start-date=2008/08/08 --type=all              \
       --proto=17 --aport=1434 --pass=stdout               \
  | rwcut --python-file direction.py                       \
       --fields internal-ip,int_to_ext,external-ip,3-12
```

Create integer key field for rwsort with the advanced API

The following example Python plug-in creates one new field, `lowest_port`, for use in **rwsort**. Using this field will sort records based on the lesser of the source port or destination port; for example, flows where either the source or destination port is 22 will occur before flows where either port is 25. This example shows using the Python **struct** module with multiple record attributes.

```
import struct

portpair = struct.Struct("!HH")

def lowest_port(rec):
    if rec.sport < rec.dport:
```

```
        return portpair.pack(rec.sport, rec.dport)
    else:
        return portpair.pack(rec.dport, rec.sport)

register_field("lowest_port", bin_bytes = portpair.size,
               rec_to_bin = lowest_port)
```

To use this example to sort the records in *flowrec.rw*, one saves the code to the file *sort.py* and uses it as shown:

```
$ rwsort --python-file=sort.py --fields=lowest_port        \
       flowrec.rw > outfile.rw
```

Create integer key for rwstats and rwuniq with advanced API

The following example defines two key fields for use by **rwstats** or **rwuniq**: `prefixed-sip` and `prefixed-dip`. Using these fields, the user can count flow records based on the source and/or destination IPv4 address blocks (CIDR blocks). The default CIDR prefix is 16, but it can be changed by specifying the **--prefix** switch that the example creates. This example uses the Python **struct** module to convert between the IP address and a binary string.

```
import os, struct
from silk import *

default_prefix = 16

u32 = struct.Struct("!L")

def set_mask(prefix):
    global mask
    mask = 0xFFFFFFFF
    # the value we are handed is a string
    prefix = int(prefix)
    if 0 < prefix < 32:
        mask = mask ^ (mask >> prefix)

# Convert from an IPv4Addr to a byte sequence
def cidr_to_bin(ip):
    if ip.is_ipv6():
        raise ValueError, "Does not support IPv6"
    return u32.pack(int(ip) & mask)

# Convert from a byte sequence to an IPv4Addr
def cidr_bin_to_text(string):
    (num,) = u32.unpack(string)
    return IPv4Addr(num)

register_field("prefixed-sip", column_width = 15,
               rec_to_bin = lambda rec: cidr_to_bin(rec.sip),
               bin_to_text = cidr_bin_to_text,
               bin_bytes = u32.size)
```

```
register_field("prefixed-dip", column_width = 15,
               rec_to_bin = lambda rec: cidr_to_bin(rec.dip),
               bin_to_text = cidr_bin_to_text,
               bin_bytes = u32.size)

register_switch("prefix", handler=set_mask,
                help="Set prefix for prefixed-sip/prefixed-dip fields")

set_mask(default_prefix)
```

The **lambda** expression allows one to create an anonymous function. In this code, the **lambda** function is used to pass the appropriate IP address into the **cidr_to_bin()** function. To write the code without the **lambda** would require separate functions for the source and destination IP addresses:

```
def sip_cidr_to_bin(rec):
    return cidr_to_bin(rec.sip)

def dip_cidr_to_bin(rec):
    return cidr_to_bin(rec.dip)
```

The **lambda** expression helps to simplify the code.

If the code is saved in the file *mask.py*, it can be used as follows to count the number of flow records seen in the /8 of each source IP address. The flow records are read from *flowrec.rw*. The **--ipv6-policy=ignore** switch is used to restrict processing to IPv4 addresses.

```
$ rwuniq --ipv6-policy=ignore --python-file mask.py        \
       --prefix 8 --fields prefixed-sip flowrec.rw
```

Create new average bytes value field for rwstats and rwuniq

The following example creates a new aggregate value that can be used by **rwstats** and **rwuniq**. The value is `avg-bytes`, a value that calculates the average number of bytes seen across all flows that match the key. It does this by maintaining running totals of the byte count and number of flows.

```
import struct

fmt = struct.Struct("QQ")
initial = fmt.pack(0, 0)
textsize = 15
textformat = "%%%d.2f" % textsize

# add byte and flow count from 'rec' to 'current'
def avg_bytes(rec, current):
    (total, count) = fmt.unpack(current)
    return fmt.pack(total + rec.bytes, count + 1)

# return printable representation
def avg_to_text(bin):
    (total, count) = fmt.unpack(bin)
    return textformat % (float(total) / count)
```

```
# merge two encoded values.
def avg_merge(rec1, rec2):
    (total1, count1) = fmt.unpack(rec1)
    (total2, count2) = fmt.unpack(rec2)
    return fmt.pack(total1 + total2, count1 + count2)


# compare two encoded values
def avg_compare(rec1, rec2):
    (total1, count1) = fmt.unpack(rec1)
    (total2, count2) = fmt.unpack(rec2)
    # Python 2:
    #return cmp((float(total1) / count1), (float(total2) / count2))
    # Python 3:
    avg1 = float(total1) / count1
    avg2 = float(total2) / count2
    if avg1 < avg2:
        return -1
    return avg1 > avg2


register_field("avg-bytes",
               column_width    = textsize,
               bin_bytes       = fmt.size,
               add_rec_to_bin  = avg_bytes,
               bin_to_text     = avg_to_text,
               bin_merge       = avg_merge,
               bin_compare     = avg_compare,
               initial_value   = initial)
```

To use this code, save it as *avg-bytes.py*, specify the name of the Python file in the **--python-file** switch, and list the field in the **--values** switch:

```
$ rwuniq --python-file=avg-bytes.py --fields=sip           \
       --values=avg-bytes infile.rw
```

This particular example will compute the average number of bytes per flow for each distinct source IP address in the file *infile.rw*.

Create integer key field for all tools that use fields

The following example Python plug-in file defines two fields, `sport-service` and `dport-service`. These fields convert the source port and destination port to the name of the "service" as defined in the file */etc/services*; for example, port 80 is converted to "http". This plug-in can be used by any of **rwcut**, **rwgroup**, **rwsort**, **rwstats**, or **rwuniq**.

```
import os,socket,struct


u16 = struct.Struct("!H")
```

```
# utility function to convert number to a service name,
# or to a string if no service is defined
def num_to_service(num):
    try:
        serv = socket.getservbyport(num)
    except socket.error:
        serv = "%d" % num
    return serv


# convert the encoded port to a service name
def bin_to_service(bin):
    (port,) = u16.unpack(bin)
    return num_to_service(port)


# width of service columns can be specified with the
# SERVICE_WIDTH environment variable; default is 12
col_width = int(os.getenv("SERVICE_WIDTH", 12))


register_field("sport-service", bin_bytes = u16.size,
               column_width = col_width,
               rec_to_text = lambda rec: num_to_service(rec.sport),
               rec_to_bin = lambda rec: u16.pack(rec.sport),
               bin_to_text = bin_to_service)


register_field("dport-service", bin_bytes = u16.size,
               column_width = col_width,
               rec_to_text = lambda rec: num_to_service(rec.dport),
               rec_to_bin = lambda rec: u16.pack(rec.dport),
               bin_to_text = bin_to_service)
```

If this file is named *service.py*, it can be used by **rwcut** to print the source port and its service:

```
$ rwcut --python-file service.py                          \
       --fields sport,sport-service flowrec.rw
```

Although the plug-in can be used with **rwsort**, the records will be sorted in the same order as the numerical source port or destination port.

```
$ rwsort --python-file service.py                         \
       --fields sport-service flowrec.rw > outfile.rw
```

When used with **rwuniq**, it can count flows, bytes, and packets indexed by the service of the destination port:

```
$ rwuniq --python-file service.py --fields dport-service   \
       --values=flows,bytes,packets flowrec.rw
```

Create human-readable fields for all tools that use fields

The following example adds two fields, `hu-bytes` and `hu-packets`, which can be used as either key fields or aggregate value fields. The example uses the formatting capabilities of netsa-python (http://tools.netsa.cert.org/netsa-python/index.html) to present the bytes and packets fields in a more human-friendly manner.

When used as a key, the `hu-bytes` field presents the value *1234567* as *1205.6Ki* or as *1234.6k* when the HUMAN_USE_BINARY environment variable is set to `False`.

When used as a key, the `hu-packets` field adds a comma (or the character specified by the HUMAN_THOUSANDS_SEP environment variable) to the display of the packets field. The value *1234567* becomes *1,234,567*.

The `hu-bytes` and `hu-packets` fields can also be used as aggregate value fields, in which case they compute the sum of the bytes and packets, respectively, and display it as for the key field.

The code for the plug-in is shown here, and an example of using the plug-in follows the code.

```
import silk, silk.plugin
import os, struct
from netsa.data.format import num_prefix, num_fixed

# Whether the use Base-2 (True) or Base-10 (False) values for
# Kibi/Mebi/Gibi/Tebi/... vs Kilo/Mega/Giga/Tera/...
use_binary = True
if (os.getenv("HUMAN_USE_BINARY")):
    if (os.getenv("HUMAN_USE_BINARY").lower() == "false"
        or os.getenv("HUMAN_USE_BINARY") == "0"):
        use_binary = False
    else:
        use_binary = True

# Character to use for Thousands separator
thousands_sep = ','
if (os.getenv("HUMAN_THOUSANDS_SEP")):
    thousands_sep = os.getenv("HUMAN_THOUSANDS_SEP")

# Number of significant digits
sig_fig=5

# Use a 64-bit number for packing the bytes or packets data
fmt = struct.Struct("Q")
initial = fmt.pack(0)

### Bytes functions
# add_rec_to_bin
def hu_ar2b_bytes(rec, current):
    global fmt
    (cur,) = fmt.unpack(current)
    return fmt.pack(cur + rec.bytes)

# rec_to_binary
```

```
def hu_r2b_bytes(rec):
    global fmt
    return fmt.pack(rec.bytes)

# bin_to_text
def hu_b2t_bytes(current):
    global use_binary, sig_fig, fmt
    (cur,) = fmt.unpack(current)
    return num_prefix(cur, use_binary=use_binary, sig_fig=sig_fig)

# rec_to_text
def hu_r2t_bytes(rec):
    global use_binary, sig_fig
    return num_prefix(rec.bytes, use_binary=use_binary, sig_fig=sig_fig)

### Packets functions
# add_rec_to_bin
def hu_ar2b_packets(rec, current):
    global fmt
    (cur,) = fmt.unpack(current)
    return fmt.pack(cur + rec.packets)

# rec_to_binary
def hu_r2b_packets(rec):
    global fmt
    return fmt.pack(rec.packets)

# bin_to_text
def hu_b2t_packets(current):
    global thousands_sep, fmt
    (cur,) = fmt.unpack(current)
    return num_fixed(cur, dec_fig=0, thousands_sep=thousands_sep)

# rec_to_text
def hu_r2t_packets(rec):
    global thousands_sep
    return num_fixed(rec.packets, dec_fig=0, thousands_sep=thousands_sep)

### Non-specific functions
# bin_compare
def hu_bin_compare(cur1, cur2):
    if (cur1 < cur2):
        return -1
    return (cur1 > cur2)

# bin_merge
def hu_bin_merge(current1, current2):
    global fmt
    (cur1,) = fmt.unpack(current1)
    (cur2,) = fmt.unpack(current2)
    return fmt.pack(cur1 + cur2)
```

```
### Register the fields
register_field("hu-bytes", column_width=10, bin_bytes=fmt.size,
               rec_to_text=hu_r2t_bytes, rec_to_bin=hu_r2b_bytes,
               bin_to_text=hu_b2t_bytes, add_rec_to_bin=hu_ar2b_bytes,
               bin_merge=hu_bin_merge, bin_compare=hu_bin_compare,
               initial_value=initial)

register_field("hu-packets", column_width=10, bin_bytes=fmt.size,
               rec_to_text=hu_r2t_packets, rec_to_bin=hu_r2b_packets,
               bin_to_text=hu_b2t_packets, add_rec_to_bin=hu_ar2b_packets,
               bin_merge=hu_bin_merge, bin_compare=hu_bin_compare,
               initial_value=initial)
```

This shows an example of the plug-in's invocation and output when the code below is stored in the file *human.py*.

```
$ rwstats --count=5 --no-percent --python-file=human.py    \
       --fields=proto,hu-bytes,hu-packets                  \
       --values=records,hu-bytes,hu-packets data.rw
INPUT: 501876 Records for 305417 Bins and 501876 Total Records
OUTPUT: Top 5 Bins by Records
pro|  hu-bytes|hu-packets|   Records|  hu-bytes|hu-packets|
 17|       328|         1|     15922|    4.98Mi|    15,922|
 17|      76.0|         1|     15482|    1.12Mi|    15,482|
  1|       840|        10|      5895|    4.72Mi|    58,950|
 17|      68.0|         1|      4249|     282Ki|     4,249|
 17|      67.0|         1|      4203|     275Ki|     4,203|
```

Identifying SMTP Servers

To demonstrate the use of **--python-file** in **rwfilter(1)**, we walk through a Python plug-in script that evaluates the behavior of a set of IP addresses and determines if the host is likely to be an SMTP server or relay. We expect (based on traffic studies) that more than 85% of a legitimate SMTP server's activity is devoted to sending or providing mail. If we find that the host exhibits this behavior, we include the IP address in a set called *SMTP.set*. Regardless of if the IP address is included in the set, we pass all records that appear to be legitimate mail flows.

We run the rwfilter command as follows:

```
$ rwfilter --start-date=2008/4/21 --end-date=2008/4/21       \
       --type=out,outweb --sipset=possible_SMTP_servers.set  \
       --python-file=SMTP.py --print-statistics
```

This command first collects all records of type `out` and `outweb` that have a start date on April 21, 2008. Since there are no additional command line options to filter records, all records are passed to the `rwfilter(rec)` function in *SMTP.py*. `rec` is an instance of the object `RWRec`, which represent the record being passed.

The function `rwfilter(rec)` in *SMTP.py* begins by importing the global variable `counts` and `smtpports`. `counts` is a dictionary indexed by source IP address and contains an array of size two, where the first element is the total number of bytes that the IP address has transferred and the second element is the number of bytes that the source address has transferred that are likely to be related to mail delivery.

Using the source IP address from the record, the function retrieves the current byte counts from the `counts` dictionary. If this is the first occurrence of the IP address, a new entry is added. The function then adds the byte count of this record to the total byte count and determines if the record is a mail delivery message. If it is a mail message, the function adds the bytes to the total of bytes transferred as mail and returns **True**. Otherwise, a value of **False** is returned.

After **rwfilter** processes all records it calls the `finalize()` function, which evaluates the collection of IP addresses. If the percentage of bytes that the host transferred in mail operations is greater than 85% of the total bytes transferred, the IP address is added to a final set of SMTP servers. The final set of SMTP servers is then saved to the *SMTP.set* file, and **rwfilter** exits.

```
from silk import *

# Collection of ports commonly used by SMTP servers
smtpports = set([25, 109, 110, 143, 220, 273, 993, 995, 113])

# Minimum percentage of mail traffic before being considered a mail server
threshold = 0.85

# Collection of byte counts
counts = dict()

# This function is run over all records.
# Input:  An instance of the RWRec class representing the
#         current record being processesed
# Output: True or false value indicating if the record passes
#         or fails the filter
def rwfilter(rec):
    # Import the global variables needed for processing the record
    global smtpports, counts

    # Pull data from the record
    sip = rec.sip
    bytes = rec.bytes

    # Get a reference to the current data on the IP address in question
    data = counts.setdefault(sip, [0, 0])

    # Update the total byte count for the IP address
    data[0] += bytes

    # Is the flow mail related?  If so add the byte count to the mail bytes
    if (rec.protocol == 6 and rec.sport in smtpports and
        rec.packets > 3 and rec.bytes > 120):
        data[1] += bytes
        return True

    # If not mail related, fail the record
    return False
```

```
# This is run after all records have been processed
def finalize():
    # Import the global vriables needed to evaluate the results
    global counts, threshold

    # The IP set of SMTP servers
    smtp = IPSet()

    # Iterate through all of the IP addresses.
    for ip, data in counts.iteritems():
        if (float(data[1]) / data[0]) > threshold:
            smtp.add(ip)

    # Generate the IPset of all smtp servers.
    smtp.save('smtp.set')

# Register these functions with rwfilter
register_filter(rwfilter, finalize=finalize)
```

UPGRADING LEGACY PLUGINS

Some functions were marked as deprecated in SiLK 2.0, and have been removed in SiLK 3.0.

Prior to SiLK 2.0, the **register_field()** function was called **register_plugin_field()**, and it had the following signature:

register_plugin_field(*field_name***,** [**bin_len=***bin_bytes_value***,**] [**bin_to_text=***bin_to_text_func***,**] [**text_len=***column_width_value***,**] [**rec_to_bin=***rec_to_bin_func***,**] [**rec_to_text=***rec_to_text_func*])

To convert from **register_plugin_field** to **register_field**, change **text_len** to **column_width**, and change **bin_len** to **bin_bytes**. (Even older code may use **field_len**; this should be changed to **column_width** as well.)

The **register_filter()** function was introduced in SiLK 2.0. In versions of SiLK prior to SiLK 3.0, when **rwfilter** was invoked with **--python-file** and the named Python file did not call **register_filter()**, **rwfilter** would search the Python input for functions named **rwfilter()** and **finalize()**. If it found the **rwfilter()** function, **rwfilter** would act as if the file contained:

```
register_filter(rwfilter, finalize=finalize)
```

To update your pre-SiLK 2.0 **rwfilter** plug-ins, simply add the above line to your Python file.

ENVIRONMENT

PYTHONPATH

This environment variable is used by Python to locate modules. When **--python-file** or **--python-expr** is specified, the application must load the Python files that comprise the PySiLK package, such as *silk/__init__.py*. If this *silk/* directory is located outside Python's normal search path (for example, in the SiLK installation tree), it may be necessary to set or modify the PYTHONPATH environment variable to include the parent directory of *silk/* so that Python can find the PySiLK module.

PYTHONVERBOSE

If the SiLK Python extension or plug-in fails to load, setting this environment variable to a non-empty string may help you debug the issue.

SILK_PYTHON_TRACEBACK

When set, Python plug-ins will output trace back information regarding Python errors to the standard error.

SEE ALSO

pysilk(3), **rwfilter(1)**, **rwcut(1)**, **rwgroup(1)**, **rwsort(1)**, **rwstats(1)**, **rwuniq(1)**, **pmapfilter(3)**, **silk(7)**, **python(1)**, http://docs.python.org/

5

SiLK File Formats

The formats of some SiLK files are described in this section.

sensor.conf

Sensor Configuration file for rwflowpack and flowcap

DESCRIPTION

As part of collecting flow data, the **rwflowpack(8)** and **flowcap(8)** daemons need to know what type of data they are collecting and how to collect it (e.g., listen on 10000/udp for NetFlow v5; listen on 4740/tcp for IPFIX). In addition, the **rwflowpack** daemon needs information on how to categorize the flow: for example, to label the flows collected at a border router as incoming or outgoing. The Sensor Configuration file, *sensor.conf*, contains this information, and this manual page describes the syntax of the file (see SYNTAX below) and provides some example configurations (see EXAMPLES).

The *sensor.conf* file may have any name, and it may reside in any location. The name and location of the file is specified by the **--sensor-configuration** switch to **rwflowpack** and **flowcap**.

The Sensor Configuration File defines the following concepts:

probe

> A probe specifies a source for flow data. The source could be a port on which **flowcap** or **rwflowpack** collects NetFlow or IPFIX data from a flow generator such as a router or the **yaf** software (http://tools.netsa.cert.org/yaf/). In **rwflowpack**, the source can be a directory to periodically poll for files containing NetFlow v5 PDUs, IPFIX records, or SiLK Flow records. When defining a probe, you must specify a unique name for the probe and the probe's type.

group

> A group is a named list that contains one of the following: CIDR blocks, the names of IPset files, or integers representing SNMP interfaces or VLAN identifiers. The use of groups is optional; the primary purpose of a group is to allow the administrator to specify a commonly used list (such as the IP space of the network being monitored) in a single location.

sensor

> A sensor represents a logical collection point for the purposes of analysis. The sensor contains configuration values that **rwflowpack** uses to categorize each flow record depending on how the record moves between networks at the collection point. Since the sensors and the categories (known as *flowtypes* or as *class/type* pairs) are also used for analysis, they are defined in the *Site* Configuration file, described in **silk.conf(5)**. The Sensor Configuration file maps sensors to probes and specifies the rules required to categorize the data. Usually one sensor corresponds to one probe; however, a sensor may be comprised of multiple probes, or the flow data collected at a single probe may be handled by multiple sensors.

The next section of this manual page describes the syntax of the *sensor.conf* file.

Using the syntax to configure a sensor requires knowledge of the packing logic that **rwflowpack** is using. The *packing logic* is the set of rules that **rwflowpack** uses to assign a flowtype to each record it processes. The default packing logic is for the **twoway** site, which is described in the **packlogic-twoway(3)** manual page. Additional packing logic rules are available (e.g., **packlogic-generic(3)**).

The last major section of this document is EXAMPLES where several common configurations are shown. These examples assume **rwflowpack** is using the packing logic from the **twoway** site.

SYNTAX

When parsing the Sensor Configuration file, blank lines are ignored. At any location in a line, the character `#` indicates the beginning of a comment, which continues to the end of the line. These comments are ignored.

All other lines begin with optional leading whitespace, a command name, and one or more arguments to the command. Command names are a sequence of non-whitespace characters, not including the character `#`. Arguments are textual atoms: any sequence of non-whitespace, non-`#` characters, including numerals and punctuation.

There are four contexts for commands: top-level, probe block, group block, and sensor block. The probe block, group block, and sensor block contexts are used to describe individual features of probes, groups, and sensors, respectively.

The valid commands for each context are described below.

Top-Level Commands

In addition to the commands to begin a probe, group, or sensor block, the top-level context supports the following command:

include "*path*"

The **include** command is used to include the contents of another file whose location is *path*. This may be used to separate large configurations into logical units. The argument to **include** must be a double-quoted string.

Probe Block

With the exception of the **probe** command, the commands listed below are accepted within the probe context. Within a probe block, one and only one of the following must be specified: **listen-on-port** to listen on a network socket, **poll-directory** to poll a directory for files, **read-from-file** to read a single file, or **listen-on-unix-socket** to listen on a UNIX domain socket. These commands are described below.

probe *probe-name probe-type*

The **probe** command is used in the top-level context to begin a new probe block which continues to the **end probe** command. The arguments to the **probe** command are the name of the probe being defined and the probe type. The *probe-name* must be unique among all probes. It must begin with a letter, and it may not contain whitespace characters or the slash character (`/`). When a probe is associated with a single sensor, it is good practice to give the probe the same name as the sensor. The *probe-type* must be one of the following:

netflow-v5

This probe processes NetFlow v5 protocol data units (PDU) that the daemon reads from a UDP port or from a file. NetFlow may be generated by a router or by software that reads packet capture (**pcap(3)**) data and generates NetFlow v5 records.

netflow

This is an alias for **netflow-v5** for backwards compatibility. This alias is deprecated, and it may be removed in a future release.

ipfix

An IPFIX probe processes Internal Protocol Flow Information eXchange records that the daemon reads over the network from an IPFIX source such as **yaf(1)**. An IPFIX probe can also poll a directory for files generated by the **yaf** program. To support IPFIX probes, SiLK must be built with support for the external library libfixbuf, version 1.7.0 or later. Both **yaf** and libfixbuf are available from http://tools.netsa.cert.org/.

netflow-v9

This probe processes NetFlow v9 protocol data units (PDU) that the daemon reads from a UDP port from a router. To support NetFlow v9 probes, SiLK must be built with support for the external library libfixbuf, version 1.7.0 or later.

sflow

This probe processes sFlow v5 records that the daemon reads from a UDP port. To support sFlow probes, SiLK must be built with support for the external library libfixbuf, version 1.7.0 or later. *Since SiLK 3.9.0.*

silk

A SiLK probe processes the records contained in SiLK Flow files created by previous invocations of **rwflowpack**. The flows will be completely re-packed, as if they were just received over the network. The sensor and flowtype values in each flow will be ignored. Note that SiLK usually removes the SNMP interfaces from its flow records, and it is likely that you will be unable to use the SNMP interfaces to pack the flows.

end probe

The **end probe** command ends the definition of a probe. Following an **end probe** command, top-level commands are again accepted.

listen-on-port *port*

This command configures the probe to accept flow records over the network, and *port* specifies the network port number where the probe should listen for flow data. The **protocol** command is required when **listen-on-port** is specified, and the **listen-as-host** and **accept-from-host** commands are optional. Multiple probes may use the same value for *port* as long as the probes are the same type and the **accept-from-host** command is specified in each probe block. Probes of different types may not bind to the same port, meaning the combination of the following three values must be different: **listen-on-port**, **protocol**, and **listen-as-host**. When listening to IPFIX data from **yaf**, this is the value specified to **yaf**'s **--ipfix-port** switch. When listening to NetFlow from a Cisco router, this is the *port* that was specified to the Cisco IOS command

```
ip flow-export [ip-address] [port]
```

protocol { tcp | udp }

This command, required when **listen-on-port** is given, specifies whether the port is a `tcp` or `udp` port. IPFIX probes support both types; the only permitted value for all other probe types is `udp`. When listening to IPFIX data from **yaf**, this is the value specified to **yaf**'s **--ipfix** switch.

accept-from-host ***host-name*** [***host-name...***]

This optional command specifies the hosts that are allowed to connect to the port where the probe is listening. The argument is a list of IP addresses and/or hostnames separated by whitespace and/or a comma. When this command is not present, any host may connect. The command may only be specified when the **listen-on-port** command is also present. When multiple probes use the same **listen-on-port**, **protocol**, and **listen-as-host** values, the **accept-from-host** switch must be used so that **rwflowpack** may assign incoming records to a specified probe. When listening for NetFlow,

this attribute would be the IP address of the router as seen from the machine running **rwflowpack** or **flowcap**. (Prior to SiLK 3.10.1, the **accept-from-host** command accepted only a single argument.)

listen-as-host ***host-name***

This optional command is used on a multi-homed machine to specify the address the probe should listen on (bind(2) to). Its value is the name of the host or its IP address. If not present, the program will listen on all the machine's addresses. The command may only be specified when the **listen-on-port** command is also present. For listening to NetFlow, the value would be the *ip-address* that was specified to the Cisco IOS command

```
ip flow-export [ip-address] [port]
```

listen-on-unix-socket ***path-to-unix-socket***

The value contains the path name to a UNIX domain socket where the flow generator writes its data. The parent directory of *path-to-unix-socket* must exist. Multiple probes may not use the same *path-to-unix-socket*.

poll-directory ***directory-path***

When this command is given, **rwflowpack** will periodically poll the *directory-path* to look for files to process. **flowcap** will exit with an error if you attempt to use probes that contain this command since **flowcap** does not support reading data from files. Multiple probes may not use the same *directory-path*. When polling the directory, zero length files and files whose name begin with a dot (`.`) are ignored. This command may be used with the following probe types:

- For SiLK probes, each file must be a valid SiLK Flow file.
- IPFIX probes can process files created by the **yaf** program.
- A NetFlow v5 probe will process files containing NetFlow v5 PDUs. The format of these files is specified in the description of the **read-from-file** command.

read-from-file ***dummy-value***

When this command is given, **rwflowpack** will read NetFlow v5 records from the file specified by the **--netflow-file** command line switch. The value to the **read-from-file** command is completely ignored, and we recommend you use `/dev/null` as the value. **flowcap** will exit with an error if you attempt to use probes that contain this command since **flowcap** does not support reading data from files. The format of a NetFlow v5 file is that the file's length should be an integer multiple of 1464 bytes, where 1464 is the maximum length of the NetFlow v5 PDU. Each 1464 block should contain the 24-byte NetFlow v5 header and space for thirty 48-byte flow records, even if fewer NetFlow records are valid. **rwflowpack** will accept NetFlow v5 files that have been compressed with the **gzip(1)** program.

log-flags { none | { all | bad | default | firewall-event | missing | record-timestamps | sampling | show-templa

This optional command accepts a comma- and/or space-separated list of names that specify which messages to log for this probe. If not specified, the default is **default**, which is equivalent to **bad**, **missing**, **sampling**. The possible values are:

all

Log everything.

bad

Write messages about an individual NetFlow v5 record where the packet or octet count is zero, the packet count is larger than the octet count, or the duration of the flow is larger than 45 days.

default

Enable the following values: **bad**, **missing**, **sampling**. This is the default value. *Since SiLK 3.10.0.* (Prior to SiLK 3.10.0, **all** was the default.)

firewall-event

When the `firewall-event` quirks flag is set and the probe is processing NetFlow v9 or IPFIX, write messages about records that are ignored because the firewall event information element on the record is something other than flow deleted or flow denied. *Since SiLK 3.8.1.*

missing

Examine the sequence numbers of NetFlow v5 packets and write messages about missing and out-of-sequence packets. (You may suppress messages regarding out-of-sequence NetFlow v9 or IPFIX packets for **all** probes by setting the SILK_LIBFIXBUF_SUPPRESS_WARNINGS environment variable.)

none

Log nothing. It is an error to combine this value with any other.

record-timestamps

Log the timestamps that appear on each record. This produces a lot of output, and it is primarily used for debugging. *Since SiLK 3.10.0.*

sampling

Write messages constructed by parsing the NetFlow v9 Options Templates that specify the sampling algorithm (when samplingAlgorithm and samplingInterval IEs are present) or flow sampler mode (when flowSamplerMode and flowSamplerRandomInterval IEs are present). *Since SiLK 3.8.0.*

show-templates

Write messages to the log file describing each IPFIX template that is read by this file-base or TCP probe. (UDP probes must still rely on the SILK_IPFIX_PRINT_TEMPLATES environment variable.) The message contains embedded new lines, with the template ID and domain on the first line, and each of the template's elements on the following lines. Each element is described by its name, its IE number with the private enterprise number if any, and its length in the template. Scope elements in options templates are marked. The format is that described in Section 10.2 of RFC7013. *Since SiLK 3.19.0.*

interface-values { snmp | vlan }

This optional command specifies the values that should be stored in the `input` and `output` fields of the SiLK Flow records that are read from the probe. If this command is not given, the default is **snmp**. Note that NetFlow v5 probes only support **snmp**.

snmp

Store the index of the network interface card (*ifIndex*) where the flows entered and left the router, respectively.

vlan

Store the VLAN identifier for the source and destination networks, respectively. If only one VLAN ID is available, `input` is set to that value and `output` is set to 0.

This setting does not affect whether **rwflowpack(8)** stores the `input` and `output` fields to its output files. Storage of those fields is controlled by **rwflowpack**'s **--pack-interfaces** switch.

quirks { none | { firewall-event | missing-ips | nf9-out-is-reverse | nf9-sysuptime-seconds | zero-packets ... } }

This optional command is used to indicate that special (or quirky) handling of the incoming data is desired. The value **none** disables all quirks, and that is the default setting. If the value is not **none**, it may be a list of one or more of the values specified below separated by commas and/or spaces. *Since SiLK 3.8.0.*

firewall-event

Enable checking for *firewall event* information elements (IEs) when processing IPFIX or NetFlow v9 flow records. This quirk must be enabled when collecting data from a Cisco ASA. The IPFIX firewallEvent IE is 233. The Cisco elements are NF_F_FW_EVENT (IE 40005) and NF_F_FW_EXT_EVENT (IE 33002). When this quirk is active, firewall events that match the value 2 (flow deleted) are categorized as normal flows, firewall events that match the value 3 (flow denied) are usually put into one of non-routed types (e.g., **innull**, **outnull**, see **packlogic-twoway(3)** and **packlogic-generic(3)** for details), and all other firewall events values are dropped. (Note that a log message is generated for these dropped records; to suppress these messages, use the **log-flags** command.) When this quirk is not provided, SiLK handles these records normally, which may result in duplicate flow records. (Prior to SiLK 3.8, SiLK dropped all flow records that contained a firewall event IE.) *Since SiLK 3.8.0.*

missing-ips

Store a flow record even when the record's NetFlow v9/IPFIX template does not contain IP addresses. One change in SiLK 3.8.0 was to ignore flow records that do not have a source and/or destination IP address; this quirk allows one to undo the effect of that change. *Since SiLK 3.8.1.*

nf9-out-is-reverse

Change handling of the OUT_BYTES and OUT_PKTS information elements to match that in libfixbuf prior to 1.8.0. Specifically, treat information elements 23 and 24 (OUT_BYTES and OUT_PKTS in RFC3954) as reverseOctetDeltaCount and reversePacketDeltaCount, respectively. Starting with libfixbuf-1.8.0, those NetFlow v9 elements are mapped to postOctetDeltaCount and postPacketDeltaCount, respectively. *Since SiLK 3.17.2.*

nf9-sysuptime-seconds

Work around an issue with NetFlow v9 records created by some middleboxes (e.g., SonicWall) where the sysUpTime field in the packet header is reported in seconds instead of in milliseconds. The incorrect units cause the time stamps on flow records to be future dated. In addition, adjust the time fields on single packet flow records. *Since SiLK 3.14.0.*

none

Do not enable any quirks.

zero-packets

Enable support for flow records either that do not contain a valid packets field, such as those from the Cisco ASA series of routers, or that have an unusually large bytes-per-packet ratio. When this quirk is active, SiLK sets the packet count to 1 when the incoming IPFIX or NetFlow v9 flow record has a the packet count if 0. This quirk may modify the file format used by **rwflowpack** for IPv4 records in order to support large byte-per-packet ratios. *Since SiLK 3.8.0.*

priority ***value***

This optional command is deprecated. It exists for backwards compatibility and will be removed in the next major release.

To summarize the probe types and the input they can accept:

```
 Probe Type   Berkeley    Directory  UnixDomain    Single
               Socket      Polling     Socket       File
```

```
==========  ==========  ==========  ==========  ==========
ipfix        tcp/udp        yes
netflow-v5     udp          yes                    yes
netflow-v9     udp
sflow          udp
silk                        yes
```

Lists of CIDR Blocks, IPsets, or Integers

This subsection describes the syntax of a list of CIDR blocks, a list of IPset file names, and a list of integers. These lists are used in the sensor block and group block commands described below.

A group block (see Group Block) allows you to assign names to these lists. Once the name is defined, it may be referenced in other lists of the same type by prepending the "at" character (`@`) to the group's name.

The lists are:

cidr-block-list

A cidr-block-list (or ipblock-list) contains one or more CIDR blocks or group references that represent an address space. Adjacent values in the list may be separated by multiple whitespace (space or tab) characters and/or a single comma. When IPv4 and IPv6 addresses combined, IPv4 addresses are mapped into the ::ffff:0:0/96 netblock. For lists containing more than a few CIDR blocks, consider using an IPset list instead.

ipset-list

An ipset-list contains the path names of one or more binary IPset files or group references. To create an IPset file, use the **rwsetbuild(1)** tool. Each path name may be a double-quoted string (`"example"`); the quote characters are not necessary if the path name does not contain whitespace or any special characters (single-quote `'`, double-quote `"`, comma `,`, or pound `#`). Adjacent values in the list may be separated by multiple whitespace (space or tab) characters and/or a single comma. When multiple IPset files are specified, a new IPset is created in memory and the contents of the files are merged into it. **rwflowpack(8)** exits with an error if the IPset file does not exist or does not contain any IP addresses. *Since SiLK 3.10.0.*

interface-list

An interface-list contains one or more integers between 0 and 65535, inclusive, or group references or that represent SNMP interface indexes or VLAN identifiers. Adjacent values in the list may be separated by multiple whitespace (space or tab) characters and/or a single comma.

Group Block

The use of group blocks is optional. They are a convenience to define and give a name to a list of commonly used CIDR blocks, IPset files, or integer values that are treated as SNMP interfaces or VLAN identifiers. Groups may be used in sensor blocks (Sensor Block) as described in the descriptions for the **discard-when**, **discard-unless**, *network-name*-**ipblocks**, *network-name*-**ipsets** and *network-name*-**interfaces** commands, below.

The commands in a group definition must all be of the same type. For example, you cannot mix **ipblocks** and **ipsets** commands in a single group definition, even though both contain IP addresses.

The contents of an existing group may be added to the current group block by using a group reference after the appropriate keyword as long as both groups are the same type. A *group reference* is the name of the

group prefixed by the "at" character (`@`). When a group reference is used, the contents of the existing groups are copied into the current group.

For examples of group blocks, see Group definitions below.

The **group** command is used at top-level to begin a group definition block, and the remaining commands are accepted within the group block.

group ***group-name***

The **group** command begins a new group definition block which continues to the **end group** command. The argument to the **group** command is the name of the group being defined. The *group-name* must be unique among all groups. It must begin with a letter, and it may not contain whitespace characters or the slash character (`/`).

end group

The **end group** command ends the definition of a group. Following an **end group** command, top-level commands are again accepted.

interfaces ***interface-list***

The **interfaces** command adds integer values to a group, where each integer is treated as an SNMP interface number or VLAN identifier. An *interface-list* is a list of integers or group references as defined above (Lists of CIDR Blocks, IPsets, or Integers). The **interfaces** command may appear multiple times in a group block.

ipblocks ***cidr-block-list***

The **ipblocks** command adds CIDR block values to a group. The *cidr-block-list* is described above. The **ipblocks** command may appear multiple times in a group block.

ipsets ***ipset-list***

The **ipset** command adds the IP addresses specified in an IPset file to a group. The **ipsets** command may appear multiple times in a group block. *Since SiLK 3.10.0.*

Sensor Block

The information from the sensor block is used by **rwflowpack** to determine how to categorize a flow; that is, in which file the flow record is stored. The **packlogic-twoway(3)** manual page describes how **rwflowpack** may use the sensor blocks to determine a record's category.

When the Sensor Configuration file is used with **flowcap**, no sensors need to be defined. In fact, **flowcap** completely ignores all text inside each sensor block.

The sensor block works with the packing logic to determine where **rwflowpack** stores flow records. The packing logic plug-in specifies a list of network names, and you will refer to these networks when you configure the sensor block. Most plug-ins provide the `external`, `internal`, and `null` names, where internal refers to network being monitored, null are flows that were blocked by the router's access control list, and external is everything else.

Several of the commands described below that categorize flow records require as an argument a list of CIDR blocks, a list of IPset files, or a list of integers. The syntax of these lists is described in the Lists of CIDR Blocks, IPsets, or Integers section above.

As part of determining how to process a flow record, **rwflowpack** may check a record's source or destination IP address against a cidr-block-list or an ipset-set. Note the following:

- for a cidr-block-list, the IP address is sequentially compared to each element of the list, stopping once a match is made
- when comparing an IPv4 address to an IPv6 list, the IPv4 address is converted to IPv6 by mapping it into the ::ffff:0:0/96 prefix for purposes of the comparison
- when comparing an IPv6 address to an IPv4 list, an IPv6 address in the ::ffff:0:0/96 prefix is converted to IPv4 for purposes of the comparison and any other IPv6 address fails the comparison

As part of determining how to process a flow record, **rwflowpack** may check whether the record's `input` or `output` fields are an interface-list. Whether the `input` and `output` fields contain SNMP interfaces or VLAN identifiers is determined by the **interface-values** command in the probe block (c.f. Probe Block).

The **sensor** command is used in the top-level context to begin a sensor configuration block, and the remaining commands are accepted within the sensor block.

sensor ***sensor-name***

The **sensor** command begins a new sensor configuration block. It takes as an argument the name of the sensor being configured, and that sensor must be defined in the Site Configuration file (see **silk.conf(5)**). A sensor block is closed with the **end sensor** command. You may have multiple sensor blocks that have the same *sensor-name*.

end sensor

The **end sensor** command ends the configuration of a sensor. Following an **end sensor** command, top-level commands are again accepted.

probe-type*-probes** ***probe-name [***probe-name* ...**]

This command associates the listed probe names of the given probe type with the sensor. The probes do not have to be defined before they are used. (Note this also means that a mistyped *probe-name* will not be detected.) For example, netflow-v5-probes S1 says that S1 is a netflow-v5 probe; whenever flow data arrives on the S1 probe, the sensor associated with the probe notices that data is available and processes it. Adjacent probe names in the argument list may be separated by space or tab characters and/or a single comma.

source-network ***network-name***

This command causes the sensor to assume that all flows originated from the network named *network-name*. For example, if a sensor is associated a probe that only monitors incoming traffic, you could use `source-network external` to specify that all traffic originated from the external network.

destination-network ***network-name***

This command causes the sensor to assume that all flows were sent to the network named *network-name*.

***network-name*-ipblocks {*cidr-block-list* | remainder}**

This command specifies the IP-space that is assigned to the network named *network-name*. The value of the command can be the keyword **remainder** or a cidr-block-list as defined above. When the value is the keyword **remainder**, the IP-space for *network-name* is conceptually all IPs not assigned to other networks on this sensor. The **remainder** keyword may only appear one time within a sensor block.

***network-name*-ipsets {*ipset-list* | remainder}**

This command specifies the IP-space that is assigned to the network named *network-name*. The value of the command can be the keyword **remainder** or an ipset-list as defined above. When the value is the keyword **remainder**, the IP-space for *network-name* is conceptually all IPs not assigned to other networks on this sensor. The **remainder** keyword may only appear one time within a sensor block.

***network-name*-interfaces {*interface-list* | remainder}**

This command specifies the SNMP interface index(es) or VLAN identifiers that are assigned to the network named *network-name*. The value of the command may be the keyword **remainder** or an interface-list as defined above. When the value is the keyword **remainder**, the interface list is computed by finding all interface values not assigned to other networks on this sensor. The **remainder** keyword may only appear one time within a sensor block.

isp-ip *ip-address* [*ip-address* ...]

This optional command may be used for a sensor that processes NetFlow data. The value to the command is a list of IP addresses in dotted-decimal notation, where the IPs are the addresses of the NICs on the router. For traffic that doesn't leave the router (and thus was sent to the router's null-interface), some packing-logic plug-ins use these IPs to distinguish legitimate traffic for the router (e.g., routing protocol traffic, whose destination address would be in this list) from traffic that violated the router's access control list (ACL).

The following optional sensor block commands provide a way to filter the flow records that **rwflowpack** packs for a sensor. Each filter begins with either **discard-when** or **discard-unless**, mentions a flow record field, and ends with a list of values.

The **discard-when** command causes the sensor to ignore the flow record if the property matches any of the elements in the list. When a match is found, **rwflowpack** immediately stops processing the record for the current sensor and the flow is not packed for this sensor.

The **discard-unless** command causes the sensor to ignore the flow record *unless* the property matches one of the elements in the list. That is, the flow record is packed only if its property matches one of the values specified in the list, and, when multiple **discard-unless** commands are present, if the flow record matches the values specified in each.

For each individual property, only one of **discard-when** or **discard-unless** may be specified.

discard-when source-interfaces *interface-list*

Instructs **rwflowpack** to discard a flow record for this sensor if the value in the flow's `input` field is listed in *interface-list*. When paired with VLAN tagging (see the **interface-values** command in the probe block), this allows the administrator to discard flows that have a specific VLAN tag. The commands **discard-when source-interfaces** and **discard-unless source-interfaces** may not be specified on the same sensor, but other **discard-** commands are permitted.

discard-unless source-interfaces *interface-list*

Instructs **rwflowpack** to discard the flow record for this sensor unless the flow's `input` field is in *interface-list*. That is, the flow record may be packed only if its `input` field matches one of the values specified in *interface-list*. When paired with VLAN tagging, this allows one to discard flows that do not have a specific VLAN tag. The commands **discard-when source-interfaces** and **discard-unless source-interfaces** may not be specified on the same sensor, but other **discard-** commands are permitted.

discard-when destination-interfaces *interface-list*

Discards a flow for this sensor when the flow's `output` field matches a value in *interface-list*. May not appear in the same sensor block with **discard-unless destination-interfaces**.

discard-unless destination-interfaces *interface-list*

Discards a flow for this sensor unless the flow's `output` field matches a value in *interface-list*. May not appear in the same sensor block with **discard-when destination-interfaces**.

discard-when any-interfaces ***interface-list***

Discards a flow for this sensor when either the flow's `input` or its `output` field matches a value in *interface-list*. May not appear in the same sensor block with **discard-unless any-interfaces**.

discard-unless any-interfaces ***interface-list***

Discards a flow for this sensor unless either the flow's `input` or its `output` field matches a value in *interface-list*. May not appear in the same sensor block with **discard-unless any-interfaces**.

discard-when source-ipblocks ***cidr-block-list***

Discards a flow for this sensor when the flow's source IP address, `sIP`, matches one of the CIDR blocks in *cidr-block-list*. May not appear in the same sensor block with **discard-unless source-ipblocks**.

discard-unless source-ipblocks ***cidr-block-list***

Discards a flow for this sensor unless the flow's source IP address, `sIP`, matches one of the CIDR blocks in *cidr-block-list*. May not appear in the same sensor block with **discard-when source-ipblocks**.

discard-when destination-ipblocks ***cidr-block-list***

Discards a flow for this sensor when the flow's destination IP address, `dIP`, matches one of the CIDR blocks in *cidr-block-list*. May not appear in the same sensor block with **discard-unless destination-ipblocks**.

discard-unless destination-ipblocks ***cidr-block-list***

Discards a flow for this sensor unless the flow's destination IP address, `dIP`, matches one of the CIDR blocks in *cidr-block-list*. May not appear in the same sensor block with **discard-when destination-ipblocks**.

discard-when any-ipblocks ***cidr-block-list***

Discards a flow for this sensor when either the flow's source IP or its destination IP address matches one of the CIDR blocks in *cidr-block-list*. May not appear in the same sensor block with **discard-unless any-ipblocks**.

discard-unless any-ipblocks ***cidr-block-list***

Discards a flow for this sensor unless either the flow's source IP or its destination IP address matches one of the CIDR blocks in *cidr-block-list*. May not appear in the same sensor block with **discard-when any-ipblocks**.

discard-when source-ipsets ***ipset-list***

Discards a flow for this sensor when the flow's source IP address, `sIP`, is in one of IPset files in *ipset-list*. May not appear in the same sensor block with **discard-unless source-ipsets**. *Since SiLK 3.10.0.*

discard-unless source-ipsets ***ipset-list***

Discards a flow for this sensor unless the flow's source IP address, `sIP`, is in one of IPset files in *ipset-list*. May not appear in the same sensor block with **discard-when source-ipsets**. *Since SiLK 3.10.0.*

discard-when destination-ipsets ***ipset-list***

Discards a flow for this sensor when the flow's destination IP address, `dIP`, is in one of the IPset files in *ipset-list*. May not appear in the same sensor block with **discard-unless destination-ipsets**. *Since SiLK 3.10.0.*

discard-unless destination-ipsets *ipset-list*

Discards a flow for this sensor unless the flow's destination IP address, `dIP`, is in one of the IPset files in *ipset-list*. May not appear in the same sensor block with **discard-when destination-ipsets**. *Since SiLK 3.10.0.*

discard-when any-ipsets *ipset-list*

Discards a flow for this sensor when either the flow's source IP or its destination IP address is in one of the IPset files in *ipset-list*. May not appear in the same sensor block with **discard-unless any-ipsets**. *Since SiLK 3.10.0.*

discard-unless any-ipsets *ipset-list*

Discards a flow for this sensor unless either the flow's source IP or its destination IP address is in one of the IPset files in *ipset-list*. May not appear in the same sensor block with **discard-when any-ipsets**. *Since SiLK 3.10.0.*

EXAMPLES

All these examples assume you are using the **packlogic-twoway(3)** packing logic plug-in to **rwflowpack(8)**.

Group definitions

The following shows how to create groups that can be used in other group blocks or in certain commands within a sensor block.

```
group G01
    interfaces 1 2, 3
    interfaces 4
end group

group G02
    interfaces 5 @G01
end group

group G03
    interfaces @G02
    interfaces 6
end group

group G11
    ipblocks 192.0.2.0/27  192.0.2.32/27,  192.0.2.64/26
    ipblocks 192.0.2.128/25
end group

group G12
    ipblocks 198.51.100.0/24  @G11
end group
```

```
group G13
    ipblocks @G12
    ipblocks 203.0.113.0/24
end group

group G21
    ipsets /var/sets/ip1.set  /var/sets/ip2.set,  /var/sets/ip3.set
    ipsets /var/sets/ip4.set
end group

group G22
    ipsets /var/sets/ip5.set  @G21
end group

group G23
    ipsets @G22
    ipsets /var/sets/ip6.set
end group
```

NetFlow v5 Categorized by SNMP Interface

The following two blocks define a probe that listens on 9900/udp for NetFlow v5 from a router. The probe only accepts traffic originating from 172.16.22.22 or 172.16.33.33. The associated sensor uses the SNMP interfaces to categorize the flows, where traffic that enters the router on interface 1 and leaves on interface 8 is **in**, traffic entering on 8 and leaving on 1 is **out**, traffic from 1 to 0 is **innull**, traffic from 8 to 8 is **int2int**, etc.

```
probe S1 netflow-v5
    listen-on-port 9901
    protocol udp
    accept-from-host 172.16.22.22 172.16.33.33
end probe

sensor S1
    netflow-v5-probes S1
    external-interfaces 1
    internal-interfaces 8
    null-interfaces 0
end sensor
```

NetFlow v5 Categorized by IP Address

The probe in this example is the same as above, except the administrator has chosen to log only messages about bad packets (messages about missing packets will be ignored). The sensor is categorizing flows by the source and destination IP address in the flow record. The internal network is defined as 128.2.0.0/16, and all other IPs are defined as external. For example, HTTP traffic whose source is 128.2.0.1 and destination is google.com will be categorized as **outweb**; the reply (source of google.com and destination 128.2.0.1) will be **inweb**.

```
probe S2 netflow-v5
    listen-on-port 9902
    protocol udp
    accept-from-host 172.16.22.22 172.16.33.33
    log-flags bad                    # ignore missing pkts
end probe

sensor S2
    netflow-v5-probes S2
    internal-ipblocks 128.2.0.0/16
    external-ipblocks remainder
end sensor
```

IPFIX Categorized by IP Address

This example uses an IPFIX probe to collect the flows on port 9903/tcp, where the probe binds to address 192.168.1.92. The sensor configuration is the same as in the previous example, but a group definition is used to define the internal network.

```
probe S3 ipfix
    listen-on-port 9903
    protocol tcp
    listen-as-host 192.168.1.92
end probe

group my-network
    ipblocks 128.2.0.0/16
end group

sensor S3
    ipfix-probes S3
    internal-ipblocks @my-network
    external-ipblocks remainder
end sensor
```

IPFIX Read from Files

This example uses the same sensor configuration as above. The probe processes files that have been created by **yaf(1)** and stored in the directory */tmp/var/yaf/*.

```
probe S4 ipfix
    poll-directory /tmp/var/yaf
end probe

sensor S4
    ipfix-probes S4
    internal-ipblock 128.2.0.0/16
    external-ipblock remainder
end sensor
```

NetFlow v9 Categorized by IP Address

This example uses a NetFlow v9 probe to collect the flows on port 9905/udp, where the probe binds to address 192.168.1.92. The sensor configuration uses an IPset file to define the internal network.

```
probe S5 netflow-v9
    listen-on-port 9905
    protocol udp
    listen-as-host 192.168.1.92
end probe

sensor S5
    netflow-v9-probes S5
    internal-ipsets /var/sets/my-network.set
    external-ipsets remainder
end sensor
```

sFlow v5 Categorized by IP Address

This example uses an sFlow probe to collect the flows on port 9906/udp, where the probe binds to the IPv6 address ::1. The sensor configuration uses an IPset file to define the internal network.

```
probe S19 sflow
    listen-on-port 9906
    protocol udp
    listen-as-host ::1
end probe

sensor S19
    sflow-probes S19
    internal-ipsets /var/sets/my-network.set
    external-ipsets remainder
end sensor
```

NetFlow v9 from a Cisco ASA Router

When collecting NetFlow v9 data from a Cisco ASA (Adaptive Security Appliance), specify the **quirks** statement as shown in this example to enable special handling of the NetFlow data.

```
probe S20 netflow-v9
    listen-on-port 9988
    protocol udp
    quirks firewall-event zero-packets
end probe

sensor S20
    netflow-v9-probes S20
    internal-ipsets /var/sets/my-network.set
    external-ipsets remainder
end sensor
```

Multiple Sources Becoming One Sensor (One Port)

Consider a scenario where there are multiple input streams that need to be treated as a single sensor. For example, you use multiple routers for load-balancing but you want them treated as a single logical sensor. In this configuration, you send all the input streams to a single port, and you define a single probe listening on that port. As long as the streams have a unique source IP, the streams will be treated distinctly.

The following sensor and probe blocks accept any number of TCP-based IPFIX connections to port 9907 and any number of NetFlow v5 connections to 9908. This configuration works for all types of input as SiLK 3.4.0 when using libfixbuf-1.2.0. See the configuration in the following example for a alternate approach.

```
probe S7 ipfix
    listen-on-port 9907
    protocol tcp
end probe

sensor S7
    ipfix-probes S7
    internal-ipblocks 128.2.0.0/16
    external-ipblocks remainder
end sensor

probe S8 netflow-v5
    listen-on-port 9908
    protocol udp
    log-flags bad
end probe

sensor S8
    netflow-v5-probes S8
    internal-ipblocks 128.2.0.0/16
    external-ipblocks remainder
end sensor
```

Multiple Sources Becoming One Sensor (Multiple Ports)

Like the previous example, this example configuration causes multiple input streams to be treated as a single sensor. In this solution, each stream arrives on a separate port where it is collected by a separate probe. The sensor block combines the probes into one sensor. This type of approach works with all types of input for all releases of SiLK.

```
probe S6-p1 netflow-v9
    listen-on-port 9961
    protocol udp
end probe

probe S6-p2 netflow-v9
    listen-on-port 9962
    protocol udp
end probe
```

```
probe S6-p3 netflow-v9
    listen-on-port 9963
    protocol udp
end probe

sensor S6
    netflow-v9-probes S6-p1, S6-p2, S6-p3
    internal-ipblocks 128.2.0.0/16
    external-ipblocks remainder
end sensor
```

Multiple Sources Becoming One Sensor (Specific Directions)

Consider the case of using **yaf** on a monitor at the border of a network where all traffic entering the network arrives at the monitor on one network interface card (NIC) and all traffic leaving the network arrives at the monitor on a different NIC. Since **yaf** does not support multiple interfaces yet, you must run two **yaf** processes, one for each NIC. The sensor configuration for this monitor would list two probes, each listening on a different port, and two sensor blocks both packing to the same sensor. Each sensor block packs the traffic as incoming or outgoing depending on which probe received the traffic.

```
probe S9-in ipfix
    listen-on-port 9991
    protocol tcp
end probe

probe S9-out ipfix
    listen-on-port 9992
    protocol tcp
end probe

sensor S9
    ipfix-probes S9-in
    source-network external
    destination-network internal
end sensor

sensor S9
    ipfix-probes S9-out
    source-network internal
    destination-network external
end sensor
```

Multiple Sources to Multiple Sensors (Same Port)

Suppose your network has multiple flow generators that you wish to treat as separate sensors, but you would like to minimize the number of open ports on your firewall. To support this configuration, configure the probes to distinguish the traffic based on the source address. Specifically, create a separate probe for each sensor where the probes of the same type use the same **listen-on-port** value but different **accept-from-host** values. (Different probe types may not bind the same port; the combination of **listen-on-port**, **protocol**,

and **listen-as-host** must be unique for different probe types.) The following configuration uses a NetFlow v5 probe, which works for all versions of SiLK. A similar configuration works for any type of input as of SiLK 3.4.0 and libfixbuf-1.2.0.

```
probe S10 netflow-v5
    listen-on-port 9910
    accept-from-host 172.16.22.10
    protocol udp
end probe

probe S11 netflow-v5
    listen-on-port 9910
    accept-from-host 172.16.22.11
    protocol udp
end probe

group my-network2
    ipblocks 128.2.0.0/16
end group

sensor S10
    netflow-v5-probes S10
    internal-ipblocks @my-network2
    external-ipblocks remainder
end sensor

sensor S11
    netflow-v5-probes S11
    internal-ipblocks @my-network2
    external-ipblocks remainder
end sensor
```

Single Source Becoming Multiple Sensors

Suppose you have instrumented a single router but you wish to split the traffic into two sensors, where one part of the network (monitored by sensor S12) is defined as 128.2.0.0/17, and the other (sensor S13) as 128.2.128.0/17. Traffic between 128.2.0.1 and google.com will be assigned to sensor S12, but it will so appear as **ext2ext** traffic for sensor S13 unless you explicitly discard that traffic using the **discard-unless** command.

```
probe S12-S13 ipfix
    listen-on-port 9912
    protocol tcp
end probe

group S12-space
    ipblocks 128.2.0.0/17
end group
```

```
group S13-space
    ipblocks 128.2.128.0/17
end group

sensor S12
    ipfix-probes S12-S13
    discard-unless any-ipblock @S12-space
    internal-ipblocks @S12-space
    external-ipblocks remainder
end sensor

sensor S13
    ipfix-probes S12-S13
    discard-unless any-ipblock @S13-space
    internal-ipblocks @S13-space
    external-ipblocks remainder
end sensor
```

Discarding Flows Using VLAN Tags

You can configure **rwflowpack** to discard flows that do not have a particular VLAN tag. First, specify the **interface-values** command to instruct the probe to put the VLAN id into the fields that typically store the SNMP interfaces. On the sensor, use the **discard-unless** command to discard flows that do not have the desired VLAN tag (114 in this example). Often you will not use the VLAN tags to determine a flow's direction (category) since there is a single VLAN tag on each flow; instead, you specify the IP space of the monitored network in the sensor block. (However, see the next example.)

```
probe S14 ipfix
    listen-on-port 9914
    protocol tcp
    interface-values vlan
end probe

sensor S14
    ipfix-probes S14
    discard-unless any-interface 114
    internal-ipblocks 128.2.0.0/16
    external-ipblocks remainder
end sensor
```

Categorizing Flows Using VLAN Tags

By repeating a sensor block and using different **discard-unless** commands in each block, you may configure **rwflowpack** to categorize flow records based on VLAN tags. Suppose **yaf** is monitoring a connection where incoming flows are marked with VLAN tag 151 and outgoing flows are marked with 152. You simply discard any traffic that does not have the wanted VLAN tag, and use the **source-network** and **destination-network** commands to assign the direction to the flow. In this example, any flow record that does not have one of the expected VLAN tags has its source-network set to `null`, but since **rwflowpack** does not expect a flow record to originate from the null network, it stores the record in the **other** category for later analysis/debugging. (This example requires SiLK 3.1 or later.)

```
probe S15 ipfix
    listen-on-port 9915
    protocol tcp
    interface-values vlan
end probe


sensor S15
    # vlan ID 151 is incoming
    ipfix-probes S15
    discard-unless source-interface 151
    source-network       external
    destination-network  internal
end sensor


sensor S15
    # vlan ID 152 is outgoing
    ipfix-probes S15
    discard-unless source-interface 152
    source-network       internal
    destination-network  external
end sensor


sensor S15
    # discard flows that have known IDs
    # force unknown IDs into the "other" category
    ipfix-probes S15
    discard-when source-interface 151,152
    source-network       null
    destination-network  internal
end sensor
```

IPFIX Collected by a DAG Card

When **yaf** generates flow records from a multi-port Endace DAG card, it is possible to use the port where the traffic was seen to categorize the traffic in **rwflowpack**.

To do this, include the **--dag-interface** switch on the **yaf** command line. This switch causes **yaf** to store the DAG port where the packet was collected into the equivalent of the SNMP input field, and **yaf** sets the SNMP output field to an offset of the port, specifically the port plus 256 (0x100|port).

Assume DAG port 0 is connected to the external side of the network (so it sees incoming traffic), and assume DAG port 1 is on the internal side. For incoming traffic, **yaf** sets the input and output values to 0 and 256, respectively. For outgoing traffic, the values are 1 and 257.

The *sensor.conf* configuration file for **rwflowpack** would be:

```
probe S16-dag ipfix
    listen-on-port 9916
    protocol tcp
end probe
```

```
sensor S16
    ipfix-probes S16-dag
    external-interface 0,257
    internal-interface 1,256
end sensor
```

When **rwflowpack** processes the IPFIX flow records, it treats flow records having an input of 0 and an output of 256 as traffic moving from an external interface to an internal interface, and **rwflowpack** packs those records as incoming. Similarly for the outgoing flow records.

Repacking of SiLK Flows by IP Address

A probe whose type is `silk` must get its flows by polling a directory of SiLK Flow files. The flows can be re-categorized based on the IP addresses or based on the SNMP interfaces (beware: often the SNMP interface values are 0 in SiLK Flow data). In this example, the files in the directory */var/tmp/old-data/* are processed. The internal network is defined as 128.2.0.0/16, and all other IPs are defined as external.

```
probe S17 silk
    poll-directory /var/tmp/old-data
end probe

sensor S17
    silk-probes S17
    internal-ipblock 128.2.0.0/16
    external-ipblock remainder
end sensor
```

NetFlow From a File Categorized by SNMP Interfaces

Instead of listening on a UDP port for NetFlow traffic, you can configure the probe to process a single file containing NetFlow v5 PDUs. This example assumes you are running **rwflowpack** with the switches **--input-mode=file --netflow-file**=*FILENAME*. The **--netflow-file** switch overrides the **read-from-file** command on the probe. **rwflowpack** will exit once it processes that single file.

```
probe S18 netflow-v5
    log-flags bad                   # ignore missing pkts
    read-from-file /dev/null        # use --netflow-file=<file>
end probe

sensor S18
    netflow-v5-probes S18
    external-interface 182
    internal-interface 189
    null-interface 0
end sensor
```

SEE ALSO

rwflowpack(8), **flowcap(8)**, **packlogic-twoway(3)**, **packlogic-generic(3)**, **rwsetbuild(1)**, **silk.conf(5)**, **silk(7)**, *SiLK Installation Handbook*, **pcap(3)**, **yaf(1)**, **gzip(1)**

NOTES

Support for using double-quoted strings for IPset path names was added in SiLK 3.17.2.

The accept-from-host command began to accept a list of arguments in SiLK 3.10.1.

SiLK 3.10.0 added IPset file support to the **group** block and to some commands in the **sensor** block.

Support for collecting sFlow records was added in SiLK 3.9.0.

The **quirks** command was introduced in SiLK 3.8.0.

silk.conf

SiLK site configuration file

DESCRIPTION

The *silk.conf* SiLK site configuration file is used to associate symbolic names with flow collection information stored in SiLK Flow records.

In addition to the information contained in the NetFlow or IPFIX flow record (e.g., source and destination addresses and ports, IP protocol, time stamps, data volume), every SiLK Flow record has two additional pieces of information that is added when **rwflowpack(8)** converts the NetFlow or IPFIX record to the SiLK format:

- The *sensor* typically denotes the location where the flow data was collected; e.g., an organization that is instrumenting its border routers would create a sensor to represent each router. Each sensor has a unique name and numeric ID.
- The *flowtype* represents information about how the flow was routed (e.g., as incoming or outgoing) or other information about the flow (e.g., web or non-web). The packing process categorizes each flow into a flowtype. Each flowtype has a unique name and numeric ID.

Note that the binary form of SiLK flow records represent the sensor and flowtype by their numeric IDs, not by their names.

For historic reasons, one rarely speaks of the flowtype of a SiLK Flow record, but instead refers to its *class* and *type*. Every flowtype maps to a unique class/type pair. The classes and types have names only; they do not have numeric IDs. Note that *flowtype* and *type* are different concepts despite the similarity of their names.

A class is generally used to represent topological features of the network with different collections of sensors, since every active sensor must belong by one or more classes. Every class must have a unique name.

A type is used to distinguish traffic within a single topological area based on some other dimension. For example, incoming and outgoing traffic is generally distinguished into different types. Web traffic is also frequently split into a separate type from normal traffic in order to partition the data better. The type names within a class must be unique, but multiple classes may have a type with the same name.

As stated above, each class/type pair maps to a unique flowtype.

The *silk.conf* file defines

- the mapping between sensor names and sensor IDs
- optional named collections of sensors called sensor-groups
- the names of the available classes
- the sensors that belong to each class
- the names of the types in each class
- the mapping from a class/type pair to a flowtype ID
- the mapping between a flowtype name and a flowtype ID

- the default class to use for **rwfilter(1)** and **rwfglob(1)** queries
- for each class, the default types to use for **rwfilter** and **rwfglob**
- the layout of the directory tree for the SiLK archive (flow data repository) relative to the root directory
- a default value for the **--packing-logic** switch to **rwflowpack(8)**

In normal usage, the *silk.conf* file will be located at the root of the SiLK data spool referenced by the SILK_DATA_ROOTDIR environment variable, or specified on the command line using the **--data-rootdir** flag. This ensures that the sensor and class definitions in the site configuration match the data in the flow records you retrieve.

If you cannot place the site configuration file in the data root directory, or the file in that location is incorrect, you can use the SILK_CONFIG_FILE environment variable to specify the location of your configuration file (including the file name). Many SiLK commands provide the **--site-config-file** switch which allows you to specify the name of the site configuration file on the command line.

By having the site configuration information outside of the SiLK tools, a single SiLK installation can be used to query different data stores (though each invocation of a command may only query one storage location).

Any additions or modifications to the *silk.conf* file will be seen by all SiLK applications upon their next invocation. There are some important things to keep in mind when modifying the *silk.conf* file:

- Once data has been collected for a sensor or a flowtype, the sensor or flowtype should never be removed or renumbered. SiLK Flow files store the sensor ID and flowtype ID as integers; removing or renumbering a sensor or flowtype breaks this mapping. In order to keep the mapping consistent, old sensor and flowtype definitions should remain indefinitely. Completely unused sensors or flowtypes may be removed, but the IDs of the remaining sensors and flowtypes must not be modified.
- The path to the files in the SiLK data store often involve the sensor name, flowtype name, class name, and/or type name. If any of those names are changed, it will be necessary to rename all the previously packed data files that have the former name as part of their path.
- If the SiLK installation at your site is distributed across multiple hosts (for example, if packing occurs on a machine separate from analysis), it is important to synchronize changes to the *silk.conf* files.
- The packing logic plug-in file, *packlogic-*.so* (e.g., **packlogic-twoway(3)**, **packlogic-generic(3)**), used by **rwflowpack(8)** checks for specific class names, type names, and flowtype names at start up, and it will exit with an error if the names it expects do not exist. In addition, it checks that the flowtype IDs it has match with those in the *silk.conf* file. When new flowtypes are added, the *packlogic-*.so* file will need to be updated if **rwflowpack** is to generate SiLK Flow records with the new flowtype.
- When **rwflowpack** reads incoming flow records, those records are associated with a sensor name as determined by the **sensor.conf(5)** file. **rwflowpack** uses the *silk.conf* file to map the sensor name to the sensor ID, and it stores the sensor ID in the SiLK records it creates. Changes to the *silk.conf* and *sensor.conf* files may need to be coordinated.

SYNTAX

When parsing the site configuration file, blank lines are ignored. Comments are also ignored and may begin at any location on a line by specifying the character `#` outside of a quoted string. The comment continues to the end of the line.

Each non-empty line begins with a command name followed by one or more arguments. Whitespace (one or more space and/or tab characters) may appear before the command and is required to separate the command and its arguments. Command names are textual atoms, while arguments may either be textual atoms or quoted strings.

- A textual atom may contain the alphanumeric characters (`A-Z`, `a-z`, `0-9`), period (`.`), hyphen (`-`), underscore (`_`), at (`@`), and forward slash (`/`). To use any other character as part of an argument, a quoted string must be used.
- Quoted strings begin and end with the double-quote character `"` which must appear on a single line. Quoted strings may contain ASCII characters and support C-style backslash-escapes. The character `#` inside a quoted string does not begin a comment.

For the commands supported by *silk.conf* and described below, unless a command explicitly states that it is used by particular applications, it should be considered used by all of the SiLK analysis tools and the packing tools **flowcap(8)**, **rwflowpack(8)**, and **rwflowappend(8)**.

There are three contexts for commands: top-level, class block, and group block contexts. The class block and group block contexts are used to describe individual features of classes and groups, while top-level commands are used to describe the entire configuration and to define sensors.

The valid commands for each context are described below.

Top-Level Commands

class *class-name*

The `class` command begins a new class block. It takes as an argument the name of the class being defined. Each class must have a unique name. A class block is closed with the `end class` command. See the Class Block Commands section below for a list of commands valid inside class blocks.

The class name must begin with a letter, must not be longer than 32 characters, and may not contain whitespace characters or these characters: forward slash (`/`), backward slash (\), double quote (`"`), single quote (`'`), and comma (`,`).

A site that does not use multiple classes should define a single class with a name like `all` or `default`.

To be valid, a configuration file must contain at least one class definition.

Example: class all

default-class *class-name*

rwfilter(1) and **rwfglob(1)** will use a default class when the user does not specify an explicit **--class** or **--flowtype**. This command specifies that default class; the class must have been created prior to this command. If more than one default class is set, the last definition encountered is used.

Example: default-class all

group *group-name*

The `group` command begins a sensor group blocks and takes as an argument the name of the group to be defined. Sensor groups are a convenient way of defining named collections of sensors for inclusion in classes. As of SiLK 3.21.0, sensor group names are accepted by the **--sensors** option of **rwfilter**, **rwfglob**, and **rwsiteinfo(1)**. A group block is closed using the `end group` command. See the Group Block Commands section below for details on valid commands within group blocks.

Example: group test-sensors

include "*file-name*"

The `include` command is used to include the contents of another file. This may be used to separate large configurations into logical units. An analyst may wish to create their own site configure file to augment the default one with additional sensor groups or to change the default-class. (Note that all sensors, classes, groups, and types must be declared before they may be referenced.)

Example: include "silk-2.conf"

packing-logic "*file-name*"

The `packing-logic` command provides a default value for the **--packing-logic** switch on **rwflowpack(8)**. The value is the path to a plug-in that **rwflowpack** loads; the plug-in provides functions that determine into which class and type a flow record will be categorized. The path specified here will be ignored when the **--packing-logic** switch is explicitly specified to **rwflowpack** or when SiLK has been configured with hard-coded packing logic.

Example: packing-logic "packlogic-twoway.so"

path-format "*format-string*"

File and directory locations relative to the *SILK_DATA_ROOTDIR* may be defined using the `path-format` command. The `path-format` is used by **rwflowpack** and **rwflowappend(8)** when writing data to the data repository, and it is used by **rwfilter** and **rwfglob** when reading or listing files in the data repository. This command takes a format string specification that supports the following `%`-conversions:

%C
: The textual class name

%d
: The day of the month as a two-digit, zero-padded number

%F
: The textual flowtype name for this class/type pair (see also `%f`)

%f
: The flowtype ID, as an unpadded number (see also `%F`)

%H
: The hour (24-hour clock) as a two-digit, zero-padded number

%m
: The month of the year as a two-digit, zero-padded number

%N
: The textual sensor name (see also `%n`)

%n
: The sensor ID, as an unpadded number (see also `%N`)

%T
: The textual type name

%x
: The default file name, which is equivalent to `%F-%N_%Y%m%d.%H`

%Y
: The year as a four-digit, zero-padded number

%%
: A literal `%` character

A `%` followed by any other character is an error.

For example, to place all spooled files directly in the data root directory, the path format `%x` could be used. To use two levels of hierarchy, the first containing the year and month, and the second containing the day and sensor name, like `2006-01/23-alpha/...`, the format would be `%Y-%m/%d-%N/%x`.

If no path format is set by the configuration file, the default path format of `%T/%Y/%m/%d/%x` is used.

All path formats are currently required to end in `/%x` so that information may be extracted from the file name. This requirement may be lifted in the future.

Example: %C/%T/%Y/%m/%d/%x

sensor ***sensor-id sensor-name***

sensor ***sensor-id sensor-name "sensor-description"***

Individual sensor definitions are created with the `sensor` command. This command creates a new sensor with the given name and numeric ID. Sensor names must begin with a letter, must not be longer than 64 characters, and may not contain whitespace characters or these characters: forward slash (`/`), backward slash (\), double quote (`"`), single quote (`'`), comma (`,`), and underscore (_).

The sensor line may may also provide an optional description of the sensor, enclosed in double quotes. The description can be used however your installation chooses to use it. The description may be viewed by specifying the `describe-sensor` field to **rwsiteinfo(1)**. (When using sensor descriptions, the file's `version` must be 2.)

It is an error to define two different sensors with the same sensor ID or the same sensor name.

A sensor may be associated with multiple classes.

NOTE: It is extremely important not to change the *sensor-id* or *sensor-name* for a given sensor once that sensor is in use. The *sensor-id* field is stored numerically in SiLK data files, and the *sensor-name* field is used to construct file names within the data root directory.

Example: sensor 0 S001

Example: sensor 0 S001 "Primary connection to ISP"

version ***version-number***

The `version` command declares that this configuration file conforms to a given version of the configuration file format. If the tools do not support this version of the configuration file, they will report an error. Currently, versions `1` and `2` of the format is defined, where version 2 indicates that sensor descriptions are present.

It is a recommended practice to include the version number at the beginning of all configuration files for compatibility with future versions.

Example: version 1

Class Block Commands

As stated above, a class block begins with the command `class` followed by the name of the class. Each class must have a unique name not longer than 32 characters.

The commands inside a class block define the sensors associated with it, the class's types, the mapping from the class/type pair to the flowtype name and flowtype ID, and the default types for the class.

end class

The `end class` command ends the definition of a class. Following an `end class` command, top-level commands are again accepted.

Example: end class

default-types ***type-name ...***

When no types are specified for the `rwfilter` or `rwfglob` commands, the default set of types for the selected class is used. Each of the types listed in this command is included as a default type of the class.

Example: default-types in inweb

sensors ***sensor-name-or-group-ref ...***

The `sensors` command is used to associate sensors with a class. In short, to declare that these sensors have data for this class. Each item in the list must be either the name of a sensor or the name of a sensor group preceded by an at (`@`) character. (If the group name is a quoted string, include the `@` within the quote characters.) When you add a sensor group, it is equivalent to adding each individual sensor in that group to the class; nothing records that the group has been added to the class. Since no record is kept, adding sensors to the group after the class block does not add those sensors to this class.

Example: sensors my-sensor-1 my-sensor-2 @my-group-1

type ***flowtype-id type-name*** [***flowtype-name***]

The `type` command defines a type name within the current class and it specifies the flowtype ID to use for that class/type pair. In addition, the `type` command may specify a flowtype name. The flowtype ID and flowtype name must be unique across the entire *silk.conf* file (and any included files). If a flowtype name is not specified, a default flowtype name is constructed by concatenating the name of the class and the name of the type. (e.g. the type `in` in the class `all` would have a flowtype name of `allin`.) Within a class, each type must have a unique name, but multiple classes may use the same type name. The type name and flowtype name must begin with a letter, must not be longer than 32 characters, and may not contain whitespace characters or these characters: forward slash (`/`), backward slash (\), double quote (`"`), single quote (`'`), and comma (`,`).

As with sensors, it is important to be careful when renumbering flowtype IDs or renaming types or flowtypes because the numeric IDs are stored in data files, and the textual names are used as portions of file and path names.

Example: type 0 in

Example: type 1 out out

Group Block Commands

A group block is a convenience used to define a collection of sensors. These collections may be used when adding the same group of sensors to multiple classes. As of SiLK 3.21.0, sensor groups may also be used as arguments to the **--sensors** option of **rwfilter(1)**, **rwfglob(1)**, and **rwsiteinfo(1)**.

A sensor group block begins with the command `group` followed by the name of the group. If multiple groups blocks use the same name, the resulting group is the union of the sensors specified in all blocks. Note that uses of `@`*`GROUP-NAME`* expand only to the sensors that have been specified in any `group` *`GROUP-NAME`* blocks seen before that use.

A group may have the same name as a sensor and the *silk.conf* file can distinguish between them. However, the name only refers to the sensor when it is used in the **--sensors** switch of **rwfilter**, **rwfglob**, and **rwsiteinfo**.

end group

Close a group block by using the `end group` command. Following this command, top-level commands are again accepted.

Example: end group

sensors ***sensor-name-or-group-ref ...***

Sensors are associated with a sensor group by means of the `sensors` command within a group block. Each item in the list must be either the name of a sensor or the name of a sensor group preceded by an at (`@`) character. (If the group name is a quoted string, include the `@` within the quote characters.) When you add a sensor group named **A** to group **B**, it is equivalent to adding each individual sensor in **A** to group **B**; nothing records that **A** has been added to **B**. Since no record of is kept, adding sensors to **A** after this group block does not add those sensors to **B**. (Though another **group B** block could be opened to re-add **A** which would update **B** with those sensors.)

Example: sensors my-sensor-1 my-sensor-2 @my-group-1

SEE ALSO

rwfilter(1), **rwfglob(1)**, **rwsiteinfo(1)**, **sensor.conf(5)**, **flowcap(8)**, **rwflowpack(8)**, **packlogic-twoway(3)**, **packlogic-generic(3)**, **rwflowappend(8)**, **silk(7)**, *SiLK Installation Handbook*

7

SiLK Miscellaneous Information

Miscellaneous manual pages are grouped in this section.

SiLK

The System for Internet-Level Knowledge

DESCRIPTION

SiLK is a collection of traffic analysis tools developed by the CERT Network Situational Awareness Team (CERT NetSA) to facilitate security analysis of large networks. The SiLK tool suite supports the efficient collection, storage, and analysis of network flow data, enabling network security analysts to rapidly query large historical traffic data sets. SiLK is ideally suited for analyzing traffic on the backbone or border of a large, distributed enterprise or mid-sized ISP.

A SiLK installation consists of two categories of applications: the analysis suite and the packing system.

Analysis Suite

The SiLK analysis suite is a collection of command-line tools for processing SiLK Flow records created by the SiLK packing system. These tools read binary files containing SiLK Flow records and partition, sort, and count these records. The most important analysis tool is **rwfilter(1)**, an application for querying the central data repository for SiLK Flow records that satisfy a set of filtering options. The tools are intended to be combined in various ways to perform an analysis task. A typical analysis uses UNIX pipes and intermediate data files to share data between invocations of tools.

The tools, configuration files, and plug-in modules that make up the analysis tools are listed below, roughly grouped by functionality.

Filtering, Sorting, and Display

rwfilter(1) partitions SiLK Flow records into one or more 'pass' and/or 'fail' output streams. **rwfilter** is the primary tool for pulling flows from the data store.

silk.conf(5) is the configuration file naming the Classes, Types, and Sensors available at your installation.

rwsort(1) sorts SiLK Flow records using a user-specified key comprised of record attributes, and writes the records to the named output path or to the standard output. Users may define new key fields using plug-ins written in C or PySiLK.

rwcut(1) prints the attributes of SiLK Flow records in a delimited, columnar, human-readable format. Users may define new printable attributes using plug-ins written in C or PySiLK.

SiLK Python Extension

pysilk(3). PySiLK, the SiLK Python extension, allows one to read, manipulate, and write SiLK Flow records, IPsets, and Bags from within Python. PySiLK may be used in a stand-alone Python program or to write plug-ins for several SiLK applications. This document describes the objects, methods, and functions that PySiLK provides. The next entry describes using PySiLK from within a plug-in.

silkpython(3). The SiLK Python plug-in provides a way to use PySiLK to define new partitioning rules for **rwfilter(1)**, new key fields for **rwcut(1)**, **rwgroup(1)**, and **rwsort(1)**, and new key or value fields for **rwstats(1)** and **rwuniq(1)**.

Counting, Grouping, and Mating

rwuniq(1) bins (groups) SiLK Flow records by a user-specified key comprised of record attributes and prints the total byte, packet, and/or flow counts for each bin. **rwuniq** may also print distinct source IP

and destination IP counts. Users may define new key fields and value fields using plug-ins written in C or PySiLK.

rwcount(1) summarizes SiLK Flow records across time, producing textual output with counts of bytes, packets, and flow records for each time bin.

rwstats(1) summarizes SiLK Flow records by a user-specified key comprised of record attributes, computes values from the flow records that match each key, sorts the results by the value to generate a Top-N or Bottom-N list, and prints the results. Users may define new key fields and value fields using plug-ins written in C or PySiLK.

rwtotal(1) summarizes SiLK Flow records by a specified key and prints the sum of the byte, packet, and flow counts for flows matching the key.

rwaddrcount(1) summarizes SiLK flow records by the source or destination IP and prints the byte, packet, and flow counts for each IP.

rwgroup(1) groups SiLK flow records by a user-specified key comprised of record attributes, labels the records with a group ID that is stored in the next-hop IP field, and writes the resulting flows to the specified output path or to the standard output. **rwgroup** requires that its input is sorted.

rwmatch(1) matches (mates) records as queries and responses and marks mated records with an ID that is stored in the next-hop IP field. **rwmatch** requires that its input is sorted.

IPsets, Bags, Aggregate Bags, and Prefix Maps

An *IPset* is a data structure and a binary file format that contains a list of IP addresses where each IP appears once (a mathematical set).

A *Bag* is a data structure and a binary file format where a key is mapped to a counter (similar to a hash table or Python dictionary). The key is either a 32-bit number or an IPv6 address, and the counter is a 64-bit number. Usually the key represents an aspect of a flow record (an IP address, a port number, the protocol) and the counter is a volume (the number of flow records, the sum of the packet counts) for the flow records that match that key.

An *Aggregate Bag* is similar to a Bag except the key and/or the counter may be comprised of multiple fields. Aggregate Bags were introduced in SiLK 3.15.0.

A *prefix map* is a data structure and file format that maps *every* IP address to string. An example prefix map gives the two-letter country code for any IP address.

rwset(1) reads SiLK Flow records and generates binary IPset file(s) containing the source IP addresses or destination IP addresses seen on the flow records.

rwsetbuild(1) reads (textual) IP addresses in dotted-quad or CIDR notation from an input file or from the standard input and writes a binary IPset file.

rwsetcat(1) prints the contents of a binary IPset file as text. Additional information about the IPset file may be printed.

rwsettool(1) performs union, intersection, difference, and sampling functions on the input IPset files, generating a new IPset file.

rwsetmember(1) determines whether the IP address specified on the command line is contained in an IPset.

rwbag(1) reads SiLK Flow records and builds binary Bag(s) containing key-count pairs. An example is a Bag containing the sum of the byte counts for each source port seen on the flow records.

rwbagbuild(1) creates a binary Bag file from a binary IPset file or from a textual input file.

rwbagcat(1) prints binary Bag files as text.

rwbagtool(1) performs operations (e.g., addition, subtraction) on binary Bag files and produces a new Bag file.

rwaggbag(1) reads SiLK Flow records and builds a binary Aggregate Bag containing key-count pairs. An example is a Aggregate Bag containing the sum of the byte counts for each source port seen on the flow records. *Since SiLK 3.15.0.*

rwaggbagbuild(1) creates a binary Aggregate Bag file from a textual input file. *Since SiLK 3.15.0.*

rwaggbagcat(1) prints binary Aggregate Bag files as text. *Since SiLK 3.15.0.*

rwaggbagtool(1) performs operations (e.g., addition, subtraction) on binary Aggregate Bag files and produces a new Aggregate Bag file. *Since SiLK 3.15.0.*

rwpmapbuild(1) reads textual input and creates a binary prefix map file for use with the Address Type (**addrtype(3)**) and Prefix Map (**pmapfilter(3)**) utilities.

rwpmapcat(1) prints information about a prefix map file as text. By default, prints each IP range in the prefix map and its label.

rwpmaplookup(1) finds information about specific IP address(es) or protocol/port pair(s) in a binary prefix map file and prints the result as text.

rwipaimport(1) imports a SiLK IPset, Bag, or Prefix Map file into the IP Address Association (IPA http://tools.netsa.cert.org/ipa/) library.

rwipaexport(1) exports a set of IP addresses from the IP Address Association (IPA) library to a SiLK IPset, Bag, or Prefix Map.

IP and Port Labeling Files

addrtype(3). The Address Type file provides a way to map an IPv4 address to an integer denoting the IP as internal, external, or non-routable.

ccfilter(3). The Country Code file provides a mapping from an IP address to two-letter, lowercase abbreviation of the country what that IP address is located. The abbreviations used by the Country Code utility are those defined by ISO 3166-1 (see for example https://www.iso.org/iso-3166-country-codes.html or https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2).

pmapfilter(3). Prefix map files provide a way to map field values to string labels based on a user-defined map file. The map file is created by **rwpmapbuild(1)**.

Run Time Plug-Ins

To use most of these plug-ins, the plug-in must be explicitly loaded into an application by using the application's **--plugin** switch and giving the plug-in's library name or path as the argument. For a plug-in named *NAME*, the library is typically named *NAME.so*.

app-mismatch(3). The application-mismatch plug-in helps to find services running on unusual or non-typical ports by causing **rwfilter(1)** to only pass a flow record when the record's application field is non-zero and its value is different than that in the source port and destination port fields.

conficker-c(3). The conficker-c plug-in was written in March 2009 to detect traffic that matches the signature of the .C variant of the Conficker worm.

cutmatch(3). The cutmatch plug-in creates a field in **rwcut(1)** that provides a more user-friendly representation of the match parameter value that **rwmatch(1)** writes into a SiLK Flow record's next hop IP field.

flowkey(3). The flowkey plug-in adds a switch and a field that computes a 32-bit hash for a flow record using the same algorithm as YAF uses for its flow key utility **getFlowKeyHash(1)**. *Since SiLK 3.15.0.*

flowrate(3). The flowrate plug-in adds switches and fields to compute packets/second, bytes/second, bytes/packet, payload-bytes, and payload-bytes/second.

int-ext-fields(3). The internal/external plug-in makes available fields containing internal and external IPs and ports (int-ip, ext-ip, int-port, and ext-port). It can be used to print, sort by, or group by the internal or external IP or port, which is useful when a single flow file contains flows in multiple directions. *Since SiLK 3.0.0.*

ipafilter(3). The IPA (IP Association) plug-in works with **rwfilter** to partition flows based on data in an IPA data store. **rwfilter** will automatically load this plug-in if it is available. The plug-in requires that SiLK be compiled with IPA support (http://tools.netsa.cert.org/ipa/).

silk-plugin(3) describes how to create and compile a new SiLK plug-in using C.

Packet and IPFIX Processing

These tools operate on packet capture (**pcap(3)**) files, IPFIX files, or files of NetFlow v5 data.

rwp2yaf2silk(1) converts a packet capture (**pcap(3)**) file---such as a file produced by **tcpdump(1)**---to a single file of SiLK Flow records. **rwp2yaf2silk** assumes that the **yaf(1)** (http://tools.netsa.cert.org/yaf/) and **rwipfix2silk(1)** commands are available on your system as it is a simple Perl wrapper around those commands.

rwipfix2silk(1) converts a stream of IPFIX (Internet Protocol Flow Information eXport) records to the SiLK Flow record format.

rwsilk2ipfix(1) converts a stream of SiLK Flow records to an IPFIX (Internet Protocol Flow Information eXport) format.

rwpcut(1) reads a packet capture file and print its contents in a textual form similar to that produced by **rwcut**.

rwpdedupe(1) detects and eliminates duplicate records from multiple packet capture input files. See also **rwdedupe(1)**.

rwpmatch(1) filters a packet capture file by writing only packets whose five-tuple and timestamp match corresponding records in a SiLK Flow file.

rwptoflow(1) reads a packet capture file and generates a SiLK Flow record for every packet.

rwpdu2silk(1) creates a stream of SiLK Flow records from a file containing NetFlow v5 PDU records.

Scan Detection

rwscan(1) attempts to detect scanning activity from SiLK Flow records. **rwscan** can produce files that may be loaded into a database and queried with **rwscanquery**.

rwscanquery(1) queries the scan database which has been populated from database load files generated by **rwscan**.

Flow File Utilities

These utility applications operate on SiLK Flow files.

rwcat(1) reads SiLK Flow records from the files named on the command line, or from the standard input when no files are provided, and writes the SiLK records to the specified output file or to the standard output if it is not connected to a terminal.

rwappend(1) appends the SiLK Flow records contained in the second through final file name arguments to the records contained in the first file name argument.

rwcombine(1) reads SiLK Flow records from files named on the command line or from the standard input. For records where the attributes field contains the *flow timed-out* flag, **rwcombine** attempts to find the record with the corresponding *continuation* flag set and combine those records into a single flow. **rwcombine** writes the results to the named output file or to the standard output. *Since SiLK 3.9.0.*

rwcompare(1) determines whether two SiLK Flow files contain the same flow records.

rwdedupe(1) reads SiLK Flow records from files named on the command line or from the standard input and writes the records to the named output path or to the standard output, removing any duplicate flow records. Note that rwdedupe will reorder the records as part of its processing.

rwnetmask(1) reads SiLK Flow records, zeroes the least significant bits of the source-, destination-, and/or next-hop-IP address(es), and writes the resulting records to the named output path or to the standard output.

rwrandomizeip(1) generates a new SiLK Flow file by substituting a pseudo-random IP address for the source and destination IP addresses in given input file.

rwrecgenerator(1) generates SiLK Flow records using a pseudo-random number generator; these records may be used to test SiLK applications. *Since SiLK 3.6.0.*

rwsplit(1) reads SiLK Flow records and generates a set of sub-files from the input. The sub-files may be limited by flow-, byte-, or packet-counts, or by unique IP count. In addition, the sub-file may contain all the flows or only a sample of them.

rwswapbytes(1) generates a new SiLK Flow file by changing the byte order of the records in a given input SiLK Flow file.

Utilities

rwfileinfo(1) prints information (type, version, etc.) about a SiLK Flow, IPset, Bag, or Prefix Map file.

rwsiteinfo(1) prints information about the sensors, classes, and types specified in the **silk.conf(5)** file.

rwtuc(1) generates SiLK flow records from textual input; the input should be in a form similar to what **rwcut(1)** generates.

rwfglob(1) prints to the standard output the list of files that **rwfilter** would normally process for a given set of file selection switches.

num2dot(1) reads delimited text from the standard input, converts integer values in the specified column(s) (default first column) to dotted-decimal IP address, and prints the result to the standard output.

rwgeoip2ccmap(1) reads the MaxMind GeoIP database and creates the country code mapping file that may be used by SiLK (see **ccfilter(3)**).

rwidsquery(1) invokes **rwfilter** to find flow records matching Snort signatures.

rwresolve(1) reads delimited text from the standard input, attempts to resolve the IP addresses in the specified column(s) to host names, and prints the result to the standard output.

silk_config(1) prints information about how SiLK was compiled; this information may be used to compile and link other files and programs against the SiLK header files and libraries.

Deprecated Tools

These tools are deprecated. Their functionality is available in other applications.

mapsid(1) maps between sensor names and sensor IDs using the values specified in the **silk.conf(5)** file. **mapsid** is deprecated as of SiLK 3.0.0, and it will be removed in the SiLK 4.0 release. This functionality is

available in **rwsiteinfo(1)**.

rwguess(8) reads a file containing NetFlow v5 PDU records and prints the SNMP interfaces that are used most often and the number of records seen for each interface. **rwguess** is deprecated as of SiLK 3.8.3, and it will be removed in the SiLK 4.0 release. Similar functionality is available using a combination of **rwpdu2silk(1)**, **rwstats(1)**, and **rwuniq(1)**.

rwip2cc(1) maps a (textual) list of IP addresses to their country code. **rwip2cc** is deprecated as of SiLK 3.0.0, and it will be removed in the SiLK 4.0 release. This functionality is available in **rwpmaplookup(1)**.

Packing System

The SiLK Packing System is comprised of daemon applications that collect flow records (IPFIX flows from **yaf(1)** or NetFlow v5 or v9 PDUs from a router), convert the records to the SiLK flow format, categorize the flows as incoming or outgoing, and write the records to their final destination in binary flat files for use by the analysis suite. Files are organized in a time-based directory hierarchy with files covering each hour at the leaves.

The tools, configuration files, and plug-ins that comprise the SiLK Packing System are:

flowcap(8) listens to flow generators (devices which produce network flow data) and stores the data in temporary files prior to transferring the files to a remote machine for processing by **rwflowpack**.

rwflowpack(8) reads flow data either directly from a flow generator or from files generated by **flowcap**, converts the data to the SiLK flow record format, categorizes the flow records according to rules loaded from a packing-logic plug-in, and writes the records either to hourly flat-files organized in a time-based directory structure or to files for transfer to a remote machine for processing by **rwflowappend**.

rwflowappend(8) watches a directory for files containing small numbers of SiLK flow records and appends those records to hourly files organized in a time-based directory tree.

rwsender(8) watches an incoming directory for files, moves the files into a processing directory, and transfers the files to one or more **rwreceiver** processes. Either **rwsender** or **rwreceiver** may act as the server (i.e., listen for incoming network connections) with the other acting as the client.

rwreceiver(8) accepts files transferred from one or more **rwsender** processes and stores them in a destination directory. Either **rwsender** or **rwreceiver** may act as the server with the other acting as the client.

rwpollexec(8) monitors a directory for incoming files and runs a user-specified command on each file.

rwpackchecker(8) reads SiLK Flow records and checks for unusual patterns that may indicate data file corruption.

sensor.conf(5) is a configuration file for sensors and probes used by **rwflowpack** and **flowcap**.

packlogic-twoway(3) is one of the plug-ins available that describe a set of rules (the packing-logic) that **rwflowpack** may use when categorizing flow records as incoming or output.

packlogic-generic(3) is one of the plug-ins available that describe a set of rules (the packing-logic) that **rwflowpack** may use when categorizing flow records as incoming or output.

ENVIRONMENT

The following environment variables affect the tools in the SiLK tool suite. The variables are listed alphabetically. (Additional environment variables that are specific to a tool are documented on the tool's manual page.)

PAGER

The applications that support paging their output use the value in this environment variable when the SILK_PAGER environment variable is not set and the application's **--pager** switch is not used.

PYTHONPATH

The Python modules and library files required to use PySiLK from **rwfilter(1)**, **rwcut(1)**, **rwsort(1)**, and **rwuniq(1)** as well as from Python itself are installed under SiLK's installation tree by default. It may be necessary to set or modify the PYTHONPATH environment variable so Python can find these files. For information on using the PySiLK module, see **silkpython(3)** as well as the *SiLK in Python* handbook.

PYTHONVERBOSE

If the SiLK Python extension or plug-in fails to load, setting this environment variable to a non-empty string may help you debug the issue.

RWRECEIVER_TLS_PASSWORD

Used by **rwreceiver(8)**, this variable specifies the password to use to decrypt the PKCS#12 file specified in the **--tls-pkcs12** switch.

RWSENDER_TLS_PASSWORD

Used by **rwsender(8)**, this variable specifies the password to use to decrypt the PKCS#12 file specified in the **--tls-pkcs12** switch.

SILK_ADDRESS_TYPES

This environment variable allows the user to specify the address types mapping file used by the fields and switches specified in the **addrtype(3)** manual page. The value may be a complete path or a file relative to SILK_PATH. See the FILES section for standard locations of this file.

SILK_CLOBBER

The SiLK tools normally refuse to overwrite existing files. Setting SILK_CLOBBER to a non-empty value (other than `0`) removes this restriction.

SILK_COMPRESSION_METHOD

For most tools that implement the **--compression-method** switch, this environment variable is used as the value for that switch when it is not provided. *Since SiLK 3.13.0.*

SILK_CONFIG_FILE

This environment variable contains the location of the site configuration file, **silk.conf(5)**. This variable has precedence over all methods of finding the site file except for the **--site-config-file** switch on an application. For additional locations where site configuration file may reside, see the FILES section.

SILK_COUNTRY_CODES

This environment variable allows the user to specify the country code mapping file used by the fields and switches specified in the **ccfilter(3)** manual page. The value may be a complete path or a file relative to SILK_PATH. See the FILES section for standard locations of this file.

SILK_DATA_ROOTDIR

This variable gives the root of directory tree where the data store of SiLK Flow files is maintained, overriding the location that is compiled into the tools (/data). The **rwfilter(1)** and **rwfglob(1)** tools use this value when selecting which flow files to process unless the user passes the **--data-rootdir** switch to the application. In addition, the SiLK tools search for the site configuration file, *silk.conf*, in this directory.

SILK_ICMP_SPORT_HANDLER

Modifies how "buggy" ICMP SiLK flow records are handled. ICMP type and code are normally encoded in the destination port field. Prior to SiLK 3.4.0, a bug existed when processing IPFIX biflow ICMP records where the type and code of the second records were stored in the source port. SiLK 3.4.0 attempts to work-around this bad encoding by modifying the buggy ICMP SiLK Flow records as they are initially read. However, the change in SiLK 3.4.0 removes a previous work-around designed to fix issues with SiLK Flow records collected prior to SiLK 0.8.0 that originated as NetFlow v5 PDUs from some types of Cisco routers. The ICMP records from these Cisco routers encoded the type and code in the source port, but the bytes were swapped from the normal encoding. When the SILK_ICMP_SPORT_HANDLER environment variable is set to `none`, all work-arounds for buggy ICMP records are disabled and the source and destination ports remain unchanged.

SILK_IPSET_RECORD_VERSION

For the IPset family of tools, this environment variable is used as the default value for the **--record-version** switch when the switch is not provided on the command line. The variable is also used by **rwbagtool(1)** and **rwaggbagtool(1)** when writing an IPset file. *Since SiLK 3.7.0.*

SILK_IPV6_POLICY

For tools that implement the **--ipv6-policy** switch, this environment variable is used as the value for that switch when it is not provided.

SILK_IP_FORMAT

For tools that implement the **--ip-format** switch, this environment variable is used as the value for that switch when it is not provided. *Since SiLK 3.11.0.*

SILK_LOGSTATS

This environment variable is currently an alias for the SILK_LOGSTATS_RWFILTER environment variable described below. The ability to log invocations may be extended to other SiLK tools in future releases.

SILK_LOGSTATS_DEBUG

If the environment variable is set to a non-empty value, **rwfilter(1)** prints messages to the standard error about the SILK_LOGSTATS value being used and either the reason why the value cannot be used or the arguments to the external program being executed.

SILK_LOGSTATS_RWFILTER

When set to a non-empty value, **rwfilter(1)** treats the value as the path to a program to execute with information about this **rwfilter** invocation. Its purpose is to provide the SiLK administrator with information on how the SiLK tool set is being used.

SILK_PAGER

When this variable is set to a non-empty string, most of the applications that produce textual output (e.g., **rwcut(1)**) automatically invoke this program to display their output a screen at a time. If set to an empty string, no paging of the output is performed. The PAGER variable is checked when this variable is not set. The **--pager** switch on an application overrides this value.

SILK_PATH

This environment variable gives the root of the directory tree where the tools are installed. As part of its search for configuration files and plug-ins, a SiLK application may use this variable. See the FILES section for details.

SILK_PLUGIN_DEBUG

When this variable is set to a non-empty value, an application that supports plug-ins prints status messages to the standard error as it tries to locate and open each of its plug-ins.

SILK_PYTHON_TRACEBACK

If a Python plug-in encounters a Python-related error and this environment variable is set to a non-empty value, the application prints the error's traceback information to the standard error.

SILK_RWFILTER_THREADS

This variable sets the number of threads **rwfilter(1)** uses while reading input files or files selected from the data store.

SILK_TEMPFILE_DEBUG

When set to 1, the library that manages temporary files for **rwcombine(1)**, **rwdedupe(1)**, **rwsort(1)**, **rwstats(1)**, and **rwuniq(1)** prints debugging messages to the standard error as it creates, re-opens, and removes temporary files.

SILK_TIMESTAMP_FORMAT

For tools that implement the **--timestamp-format** switch, this environment variable is used as the value for that switch when it is not provided. *Since SiLK 3.11.0.*

SILK_TMPDIR

This variable is used by tools that write temporary files (e.g., **rwsort(1)**) as the directory in which to store those files. When this variable is not set, the value of the TMPDIR variable is checked. The **--temp-directory** switch on an application overrides this value.

SILK_UNIQUE_DEBUG

When set to 1, the binning engine used by **rwstats(1)** and **rwuniq(1)** prints debugging messages to the standard error.

TMPDIR

When this variable is set and SILK_TMPDIR is not set, temporary files are created in this directory. The value given to an application's **--temp-directory** switch takes precedence over both variables.

TZ

When a SiLK installation is built to use the local timezone (to determine if this is the case, check the `Timezone support` value in the output from the **--version** switch on most SiLK applications), the value of the TZ environment variable determines the timezone in which timestamps are displayed and parsed. If the TZ environment variable is not set, the default timezone is used. Setting TZ to 0 or to the empty string causes timestamps to be displayed in and parsed as UTC. The value of the TZ environment variable is ignored when the SiLK installation uses UTC unless the user requests use of the local timezone via a tool's **--timestamp-format** switch. For system information on the TZ variable, see **tzset(3)** or **environ(7)**.

FILES

The following file and directory locations are used by SiLK tools. A dollar sign preceding a name enclosed in braces (e.g., `${SILK_PATH}`), refers to the value of the named environment variable.

${SILK_ADDRESS_TYPES}

${SILK_PATH}/share/silk/address_types.pmap

${SILK_PATH}/share/address_types.pmap

/usr/local/share/silk/address_types.pmap

/usr/local/share/address_types.pmap

Locations that applications check when searching for the address types mapping file used by **addrtype(3)**, **rwpmapcat(1)**, and **rwpmaplookup(1)**.

${SILK_CONFIG_FILE}

ROOT_DIRECTORY/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided. The value of *ROOT_DIRECTORY/* is the root directory of the SiLK repository; that directory may be specified by a command line switch (e.g., the **--data-rootdir** switch on **rwfilter(1)**), by the SILK_DATA_ROOTDIR environment variable, or by the default location compiled into the SiLK tools (/data).

${SILK_COUNTRY_CODES}

${SILK_PATH}/share/silk/country_codes.pmap

${SILK_PATH}/share/country_codes.pmap

/usr/local/share/silk/country_codes.pmap

/usr/local/share/country_codes.pmap

Locations that applications check when searching for the country code mapping file used by **ccfilter(3)**, **rwbag(1)**, **rwpmapcat(1)**, **rwpmaplookup(1)**, and other SiLK tools.

${SILK_DATA_ROOTDIR}/

/data/

Locations for the root directory of the data repository. Some applications provide a command line switch to specify this value (for example, the **--data-rootdir** switch on **rwfilter(1)**, **rwfglob(1)**, and **rwsiteinfo(1)**).

${SILK_PATH}/lib64/silk/

${SILK_PATH}/lib64/

${SILK_PATH}/lib/silk/

${SILK_PATH}/lib/

/usr/local/lib64/silk/

/usr/local/lib64/

/usr/local/lib/silk/

/usr/local/lib/

Directories that a SiLK application checks when attempting to load a plug-in.

${SILK_TMPDIR}/

${TMPDIR}/

/tmp/

Directory in which to create temporary files when a directory was not specified using the application's **--temp-directory** switch.

SEE ALSO

Analysts' Handbook: Using SiLK for Network Traffic Analysis, *The SiLK Reference Guide*, *SiLK Installation Handbook*, http://tools.netsa.cert.org/silk/

8

SiLK Administrator's Tools

Tools used by SiLK's packing system are described in this section.

flowcap

Capture network flow data and write it to temporary files

SYNOPSIS

```
flowcap --destination-directory=DIR_PATH
      --sensor-configuration=FILENAME [--probes=NAME[,NAME...]]
      --max-file-size=SIZE [--fc-version=NUM]
      [--timeout=TIMEOUT] [--clock-time[=OFFSET]]
      [--freespace-minimum=SIZE] [--space-maximum-percent=NUM]
      [--compression-method=COMP_METHOD]
      { --log-destination=DESTINATION
        | --log-pathname=FILE_PATH
        | --log-directory=DIR_PATH [--log-basename=LOG_BASENAME]
          [--log-post-rotate=COMMAND] }
      [--log-level=LEVEL] [--log-sysfacility=NUMBER]
      [--pidfile=FILE_PATH] [--no-chdir] [--no-daemon]
```

Help options:

```
flowcap --sensor-configuration=FILE_PATH
      { --verify-sensor-config | --verify-sensor-config=VERBOSE }

flowcap --help

flowcap --version
```

DESCRIPTION

flowcap is a daemon that collects records from routers, flow meters, and devices that produce network flow data. The records are written in the SiLK Flow record format to temporary files on disk. **flowcap** may collect NetFlow records (versions 5 or 9), IPFIX records (Internet Protocol Flow Information eXport) such as those generated by **yaf(1)**, or sFlow records.

The SiLK Flow files produced by **flowcap** are meant to be used only for temporary storage. For longer-term storage, the records should processed by the **rwflowpack(8)** daemon which assigns values to each record depending on where it was collected and writes the record to an hourly file that is stored in a directory tree.

As **flowcap** receives flow records, it stores them in files in the location specified by the **--destination-directory** switch. These files are closed on quantum boundaries, with one file per flow source per quantum. A quantum is either the amount of time represented by the **--timeout** switch or the file size represented by the **--max-file-size** switch, whichever is reached first.

To transfer the files to **rwflowpack**, **flowcap** works in tandem with the **rwsender(8)** program. **rwsender** polls the storage directory and sends the files it finds there to an **rwreceiver(8)** process for processing by **rwflowpack**.

flowcap produces files that are named *PROBE_YYYYMMDDhhmmss.XXXXXX*, where *PROBE* is the name of the probe, *YYYY* is the current year, *MM* is the current month, *DD* is the current day, *hh* is the current hour, *mm* is the current minute, *ss* is the current second, and *XXXXXX* is a random six-character string.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

For the following options, a *SIZE* may be given as an ordinary integer, or as a real number followed by a suffix `K`, `M`, `G`, or `T`, which represents the numerical value multiplied by 1,024 (kilo), 1,048,576 (mega), 1,073,741,824 (giga), and 1,099,511,627,776 (tera), respectively. For example, 1.5K represents 1,536 bytes, or one and one-half kilobytes.

General Configuration Switches

--destination-directory=*DIR_PATH*

Store aggregated packed flow files in this directory for processing by **rwsender**. *DIR_PATH* must be a complete directory path. This switch is required.

--sensor-configuration=*FILENAME*

Give the path to the configuration file that **flowcap** consults to determine how to collect flow records. The complete syntax of the configuration file is described in the **sensor.conf(5)** manual page; see also the *SiLK Installation Handbook*. This switch is required.

--probes=*NAME[,NAME...]*

Choose which of the probes described in the sensor configuration file will be used by **flowcap**. The default is to use all of the probes defined in the configuration file. This switch instructs **flowcap** to only use the specifically named probes.

--max-file-size=*SIZE*

Set the approximate maximum size of **flowcap** files to *SIZE* bytes. If a **flowcap** file exceeds *SIZE* bytes, it is closed and a new file will be created and used. In addition, before opening an output file, **flowcap** ensures there are *SIZE* bytes of free space available, and exits if there is not. This switch is required.

--timeout=*TIMEOUT*

Set the maximum duration that a **flowcap** output file remains open to *TIMEOUT* seconds. When the **--clock-time** switch is given, the first duration may be less than *TIMEOUT* seconds. If the **--timeout** switch is not specified, **flowcap** uses a default of 60 seconds.

--clock-time[=***OFFSET***]

Force **flowcap** to close its files at predictable times. When this switch is provided, **flowcap** closes its output files at *OFFSET* seconds after midnight (UTC of the current day) and at every *TIMEOUT* seconds thereafter. The default value of *OFFSET* is 0. For example, **--timeout=900 --clock-time=300** causes **flowcap** to close its output files at the 05, 20, 35, and 50 minute points in each hour. Even with this switch, files are still be closed if they exceed the size specified by **--max-file-size**.

--fc-version=*NUM*

Choose the record version for the files of IPv4 flow records that **flowcap** produces. Valid values are 2, 3, 4, and 5, and the default is 5. This switch is ignored for probes that support IPv6 addresses.

--freespace-minimum=*SIZE*

Set the minimum free space to maintain on the file system where the **--destination-directory** is located. By default, **flowcap** assumes that it has full rein over the file system on which it writes its

files. The default is to leave 1GB of free space. If **flowcap** fills this space, it exits. Flows arriving during this time will be lost. See also **--space-maximum-percent**.

--space-maximum-percent=*NUM*

Use no more than this percentage of the file system containing the **--destination-directory**. The default is to use no more than 98% of the file system. If **flowcap** fills this space, it exits. See also **--freespace-minimum**.

--compression-method=*COMP_METHOD*

Specify the compression library to use when writing output files. When no compression method is specified, flowcap files are compressed using the `best` method, regardless of the default chosen when SiLK was compiled. The valid values for *COMP_METHOD* are determined by which external libraries were found when SiLK was compiled. To see the available compression methods and the default method, use the **--help** or **--version** switch. SiLK can support the following *COMP_METHOD* values when the required libraries are available.

none

Do not compress the output using an external library.

zlib

Use the **zlib(3)** library for compressing the output. Using zlib produces the smallest output files at the cost of speed.

lzo1x

Use the *lzo1x* algorithm from the LZO real time compression library for compression. This compression provides good compression with less memory and CPU overhead.

snappy

Use the *snappy* library for compression. This compression provides good compression with less memory and CPU overhead. *Since SiLK 3.13.0.*

best

Use lzo1x if available, otherwise use snappy if available, otherwise use zlib if available.

--verify-sensor-config

--verify-sensor-config=*VERBOSE*

Verify that the syntax of the sensor configuration file is correct and then exit **flowcap**. If the file is incorrect or if it does not define any probes, an error message is printed and **flowcap** exits abnormally. If the file is correct and no argument is provided to the **--verify-sensor-config** switch, **flowcap** simply exits with status 0. If an argument (other than the empty string and `0`) is provided to the switch, the names of the probes found in the sensor configuration file are printed to the standard output, and then **flowcap** exits.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

Logging and Daemon Configuration Switches

The switches in this section determine the type of log messages that **flowcap** generates and where those messages are written.

One of the following switches are required:

--log-destination=*DESTINATION*

Specify the destination where logging messages are written. When *DESTINATION* begins with a slash `/`, it is treated as a file system path and all log messages are written to that file; there is no log rotation. When *DESTINATION* does not begin with `/`, it must be one of the following strings:

`none`
: Messages are not written anywhere.

`stdout`
: Messages are written to the standard output.

`stderr`
: Messages are written to the standard error.

`syslog`
: Messages are written using the **syslog(3)** facility.

`both`
: Messages are written to the syslog facility and to the standard error (this option is not available on all platforms).

--log-directory=*DIR_PATH*

Use *DIR_PATH* as the directory to which the log files are written; *DIR_PATH* must be a complete directory path. The log files have the form

```
DIR_PATH/LOG_BASENAME-YYYYMMDD.log
```

where *YYYYMMDD* is the current date and *LOG_BASENAME* is the application name or the value passed to the **--log-basename** switch when provided. The log files are rotated: At midnight local time, a new log is opened, the previous file is closed, and the command specified by **--log-post-rotate** is invoked on the previous day's log file. (Old log files are not removed by **flowcap**; the administrator should use another tool to remove them.) When this switch is provided, a process-ID file (PID) is also written in this directory unless the **--pidfile** switch is provided.

--log-pathname=*FILE_PATH*

Use *FILE_PATH* as the complete path to the log file. The log file is not rotated.

The following switches are optional:

--log-level=*LEVEL*

Set the severity of messages that are logged. The levels from most severe to least are: `emerg`, `alert`, `crit`, `err`, `warning`, `notice`, `info`, `debug`. The default is `info`.

--log-sysfacility=*NUMBER*

Set the facility that **syslog(3)** uses for logging messages. This switch takes a number as an argument. The default is a value that corresponds to `LOG_USER` on the system where **flowcap** is running. This switch produces an error unless **--log-destination**=syslog is specified.

--log-basename=*LOG_BASENAME*

Use *LOG_BASENAME* in place of the application name in the name of *log* files in the log directory. See the description of the **--log-directory** switch. This switch does **not** affect the name of the process-ID file.

--log-post-rotate=*COMMAND*

Run *COMMAND* on the previous day's log file after log rotation. When this switch is not specified, the previous day's log file is compressed with **gzip(1)**. When the switch is specified and *COMMAND* is the empty string, no action is taken on the log file. Each occurrence of the string `%s` in *COMMAND* is replaced with the full path to the log file, and each occurrence of `%%` is replaced with `%`. If any other character follows `%`, **flowcap** exits with an error. Specifying this switch without also using **--log-directory** is an error.

--pidfile=*FILE_PATH*

Set the complete path to the file in which **flowcap** writes its process ID (PID) when it is running as a daemon. No PID file is written when **--no-daemon** is given. When this switch is not present, no PID file is written unless the **--log-directory** switch is specified, in which case the PID is written to *LOGPATH/flowcap.pid*.

--no-chdir

Do not change directory to the root directory. When **flowcap** becomes a daemon process, it changes its current directory to the root directory so as to avoid potentially running on a mounted file system. Specifying **--no-chdir** prevents this behavior, which may be useful during debugging. The application does not change its directory when **--no-daemon** is given.

--no-daemon

Force **flowcap** to run in the foreground---it does not become a daemon process. This may be useful during debugging.

ENVIRONMENT

SILK_IPFIX_PRINT_TEMPLATES

When set to 1, **flowcap** writes messages to the log file describing each IPFIX and NetFlow v9 template it receives. This is equivalent to adding `show-templates` to the `log-flags` setting for each probe in the *sensor.conf* file. See the **sensor.conf(5)** manual page for the format of these messages. *Since SiLK 3.8.2.*

SILK_LIBFIXBUF_SUPPRESS_WARNINGS

When set to 1, **flowcap** disables all warning messages generated by libfixbuf. These warning messages include out-of-sequence packets, data records not having a corresponding template, record count discrepancies, and issues decoding list elements. *Since SiLK 3.10.0.*

FILES

sensor.conf

The location of this file must be specified by the **--sensor-configuration** switch. This file specifies **probe** blocks that tell **flowcap** how to capture data. The syntax of this file is described in the **sensor.conf(5)** manual page.

SEE ALSO

sensor.conf(5), **rwflowpack(8)**, **rwsender(8)**, **rwreceiver(8)**, **silk(7)**, **yaf(1)**, **syslog(3)**, **zlib(3)**, **gzip(1)**, *SiLK Installation Handbook*

rwflowappend

Append incremental SiLK Flow files to hourly files

SYNOPSIS

```
rwflowappend --incoming-directory=DIR_PATH --root-directory=DIR_PATH
      --error-directory=DIR_PATH [--archive-directory=DIR_PATH]
      [--flat-archive] [--post-command=COMMAND]
      [--hour-file-command=COMMAND] [--threads=N]
      [--reject-hours-past=NUM] [--reject-hours-future=NUM]
      [--no-file-locking] [--polling-interval=NUM]
      [--byte-order=ENDIAN] [--pad-header]
      [--compression-method=COMP_METHOD]
      [--site-config-file=FILENAME]
      { --log-destination=DESTINATION
        | --log-pathname=FILE_PATH
        | --log-directory=DIR_PATH [--log-basename=LOG_BASENAME]
          [--log-post-rotate=COMMAND] }
      [--log-level=LEVEL] [--log-sysfacility=NUMBER]
      [--pidfile=FILE_PATH] [--no-chdir] [--no-daemon]


rwflowappend --help


rwflowappend --version
```

DESCRIPTION

rwflowappend is a daemon that watches a directory for files that contain small numbers of SiLK Flow records---these files are called *incremental files*---as generated by **rwflowpack(8)** when it is run with **--output-mode=incremental-files** or **--output-mode=sending**. **rwflowappend** appends these SiLK Flow records to the hourly files stored in the SiLK data repository whose directory tree root is specified by the **--root-directory** switch.

The directory that **rwflowappend** watches for incremental files is specified by **--incoming-directory**. As **rwflowappend** scans this directory, it ignores a file if its size is 0 bytes or if its name begins with a dot (`.`). On each scan, if **rwflowappend** detects a file name that was not present in the previous scan, it records the name and size of the file. If the file has a different size on the next scan, the new size is recorded. Once the file has the same size on two consecutive scans, **rwflowappend** appends the file to the appropriate hourly file.

After **rwflowappend** processes an incremental file, the file is deleted unless the **--archive-directory** switch is specified, in which case the incremental file is moved to that directory or to a subdirectory of that directory depending on whether **--flat-archive** was specified. The **--post-command** switch allows a command to be executed on the incremental file after it has been moved to the archive directory.

If a fatal write error occurs (for example, the disk containing the data repository becomes full), **rwflowappend** exits. Before exiting, **rwflowappend** attempts to truncate the hourly file to the size it had when it was opened, and **rwflowappend** moves the incremental file it was reading to the directory specified by **--error-directory**.

Running **rwflowappend** separately from **rwflowpack** is used when you wish to copy the packed SiLK Flow records from the machine doing the packing to multiple machines for use by analysts. Almost any network file transport protocol may be used to move the files from the packing machine to the destination machine where **rwflowappend** is running, though we have written the **rwsender(8)** and **rwreceiver(8)** to perform this task.

Separate **rwflowpack** and **rwflowappend** processes are also recommended if you want another process (such as the Analysis Pipeline http://tools.netsa.cert.org/analysis-pipeline/) to process the SiLK Flow records as they are generated.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

General Configuration

The following switches are required:

--incoming-directory=*DIR_PATH*

Periodically scan the directory *DIR_PATH* for incremental files to append to the hourly files. As **rwflowappend** scans *DIR_PATH*, it ignores a file if its name begins with a dot (`.`) or if its size is 0 bytes. When a file is first detected, its size is recorded, and the file must have the same size for two consecutive scans before **rwflowappend** will append it to the appropriate hourly file. The interval between scans is set by **--polling-interval**. *DIR_PATH* must be a complete directory path.

--root-directory=*DIR_PATH*

Append to existing hourly files and create new hourly files in the directory tree rooted at this location. The directory tree has the same subdirectory structure as that created by **rwflowpack**. *DIR_PATH* must be a complete directory path.

--error-directory=*DIR_PATH*

Store in this directory incremental files that were NOT successfully appended to an hourly file. *DIR_PATH* must be a complete directory path.

The following switches are optional:

--archive-directory=*DIR_PATH*

Move each incremental file to *DIR_PATH* or a subdirectory of it after **rwflowappend** has successfully appended the incremental file to an hourly file. If this switch is not provided, the incremental files are deleted once they are successfully appended to an hourly file. When the **--flat-archive** switch is also provided, incremental files are moved into the top of *DIR_PATH*; when **--flat-archive** is not given, each incremental file is moved to a subdirectory of *DIR_PATH* that mirrors the path of the hourly file to which the incremental file was appended. Removing files from the archive-directory is not the job of **rwflowappend**; the system administrator should implement a separate process to clean this directory. This switch is required when the **--post-command** switch is present.

--flat-archive

When archiving incremental files via **--archive-directory**, move the files into the top of the archive-directory, not into subdirectories of it. This switch has no effect if **--archive-directory** is not also specified. This switch may be used to allow another process to watch for new files appearing in the archive-directory.

--post-command=*COMMAND*

Run *COMMAND* on each incremental file after **rwflowappend** has successfully appended it to an hourly file and moved it into the archive-directory. Each occurrence of the string `%s` in *COMMAND* is replaced with the full path to the incremental file in the archive-directory, and each occurrence of `%%` is replaced with `%`. If any other character follows `%`, **rwflowappend** exits with an error. When using this feature, the **--archive-directory** must be specified. The exit status of *COMMAND* is ignored. See also the **rwpollexec(8)** daemon.

--hour-file-command=*COMMAND*

Run *COMMAND* upon creation of a new hourly file. The string `%s` in *COMMAND* is replaced with the full path to the hourly file, and the string `%%` is replaced with `%`. If any other character follows `%`, **rwflowappend** exits with an error. The exit status of *COMMAND* is ignored.

--threads=*N*

Invoke **rwflowappend** with *N* threads reading the incremental files and writing to the repository. When this switch is not provided, **rwflowappend** runs with a single thread. *Since SiLK 3.8.2.*

--reject-hours-past=*NUM*

Reject incremental files containing records whose starting hour occurs more than this number of hours in the past relative to the current hour. Incremental files that violate this value are moved into the error directory. Times are compared using the starting hour of the flow record and the current hour. For example, flow records that start at 18:02:56 and 18:58:04 are considered 1 hour in the past whether the current time is 19:01:47 or 19:59:33. When performing live data collection, it is not uncommon to get flows one to two hours in the past due to the flow generator's active timeout (often 30 minutes) and the time to transfer the flow records through the collection system. The default is to accept all incremental files.

--reject-hours-future=*NUM*

Similar to **--reject-hours-past**, but reject incremental files containing records whose starting hour occurs more than this number of hours in the future relative to the current hour. Future dated flow records are rare, but can occur due to time drift at the sensor. The default is to accept all incremental files.

--no-file-locking

Do not use advisory write locks. Normally, **rwflowappend** obtains a write lock on an hourly file prior to writing records to it. The write lock prevents two instances of **rwflowappend** from writing to the same hourly file simultaneously. However, attempting to use a write lock on some file systems causes **rwflowappend** to exit with an error, and this switch can be use when writing data to these file systems.

--polling-interval=*NUM*

Check the incoming directory for new incremental files every *NUM* seconds. The default polling interval is 15 seconds.

--byte-order=*ENDIAN*

Set the byte order for newly created SiLK Flow files. When appending records to an existing file, the byte order of the file is maintained. The argument is one of the following:

`as-is`
: Maintain the byte order of the incremental files (i.e., the byte order specified to **rwflowpack**). This is the default.

`native`
: Use the byte order of the machine where **rwflowappend** is running.

`big`
: Use network byte order (big endian) for the flow files.

`little`
: Write the flow files in little endian format.

--compression-method=*COMP_METHOD*

Specify the compression library to use when creating new hourly files. When this switch is not given, newly created hourly files maintain the compression method used by the incremental file (i.e., the compression method specified to **rwflowpack**). When appending to an existing hourly file, the compression method of the file is maintained. The valid values for *COMP_METHOD* are determined by which external libraries were found when SiLK was compiled. To see the available compression methods and the default method. use the **--help** or **--version** switch. SiLK can support the following *COMP_METHOD* values when the required libraries are available.

none
: Do not compress the output using an external library.

zlib
: Use the **zlib(3)** library for compressing the output. Using zlib produces the smallest output files at the cost of speed.

lzo1x
: Use the *lzo1x* algorithm from the LZO real time compression library for compression. This compression provides good compression with less memory and CPU overhead.

snappy
: Use the *snappy* library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead. *Since SiLK 3.13.0.*

best
: Use lzo1x if available, otherwise use snappy if available, otherwise use zlib if available.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwflowappend** searches for the site configuration file in the locations specified in the FILES section.

Logging and Daemon Configuration

One of the following mutually-exclusive switches is required:

--log-destination=*DESTINATION*

Specify the destination where logging messages are written. When *DESTINATION* begins with a slash `/`, it is treated as a file system path and all log messages are written to that file; there is no log rotation. When *DESTINATION* does not begin with `/`, it must be one of the following strings:

`none`
: Messages are not written anywhere.

`stdout`
: Messages are written to the standard output.

`stderr`
: Messages are written to the standard error.

`syslog`
: Messages are written using the **syslog(3)** facility.

`both`
: Messages are written to the syslog facility and to the standard error (this option is not available on all platforms).

--log-directory=*DIR_PATH*

Use *DIR_PATH* as the directory where the log files are written. *DIR_PATH* must be a complete directory path. The log files have the form

```
DIR_PATH/LOG_BASENAME-YYYYMMDD.log
```

where *YYYYMMDD* is the current date and *LOG_BASENAME* is the application name or the value passed to the **--log-basename** switch when provided. The log files are rotated: At midnight local time, a new log is opened, the previous file is closed, and the command specified by **--log-post-rotate** is invoked on the previous day's log file. (Old log files are not removed by **rwflowappend**; the administrator should use another tool to remove them.) When this switch is provided, a process-ID file (PID) is also written in this directory unless the **--pidfile** switch is provided.

--log-pathname=*FILE_PATH*

Use *FILE_PATH* as the complete path to the log file. The log file is not rotated.

The following set of switches is optional:

--log-level=*LEVEL*

Set the severity of messages that are logged. The levels from most severe to least are: `emerg`, `alert`, `crit`, `err`, `warning`, `notice`, `info`, `debug`. The default is `info`.

--log-sysfacility=*NUMBER*

Set the facility that **syslog(3)** uses for logging messages. This switch takes a number as an argument. The default is a value that corresponds to `LOG_USER` on the system where **rwflowappend** is running. This switch produces an error unless **--log-destination**=syslog is specified.

--log-basename=*LOG_BASENAME*

Use *LOG_BASENAME* in place of the application name in the name of *log* files in the log directory. See the description of the **--log-directory** switch. This switch does **not** affect the name of the process-ID file.

--log-post-rotate=*COMMAND*

Run *COMMAND* on the previous day's log file after log rotation. When this switch is not specified, the previous day's log file is compressed with **gzip(1)**. When the switch is specified and *COMMAND* is the empty string, no action is taken on the log file. Each occurrence of the string `%s` in *COMMAND* is replaced with the full path to the log file, and each occurrence of `%%` is replaced with `%`. If any other character follows `%`, **rwflowappend** exits with an error. Specifying this switch without also using **--log-directory** is an error.

--pidfile=*FILE_PATH*

Set the complete path to the file in which **rwflowappend** writes its process ID (PID) when it is running as a daemon. No PID file is written when **--no-daemon** is given. When this switch is not present, no PID file is written unless the **--log-directory** switch is specified, in which case the PID is written to *LOGPATH/rwflowappend.pid*.

--no-chdir

Do not change directory to the root directory. When **rwflowappend** becomes a daemon process, it changes its current directory to the root directory so as to avoid potentially running on a mounted file system. Specifying **--no-chdir** prevents this behavior, which may be useful during debugging. The application does not change its directory when **--no-daemon** is given.

--no-daemon

Force **rwflowappend** to run in the foreground---it does not become a daemon process. This may be useful during debugging.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

ENVIRONMENT

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **rwflowappend** may use this environment variable. See the FILES section for details.

FILES

${SILK_CONFIG_FILE}

ROOT_DIRECTORY/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided, where *ROOT_DIRECTORY/* is the directory specified to the **--root-directory** switch.

SEE ALSO

rwflowpack(8), **rwreceiver(8)**, **rwsender(8)**, **rwpollexec(8)**, **rwfilter(1)**, **silk(7)**, **gzip(1)**, **syslog(3)**, **zlib(3)**, *The SiLK Installation Handbook*

NOTES

rwflowappend does not check the integrity of an hourly file before appending records to it.

Prior to SiLK 3.6.0 when a write error occurred, **rwflowappend** could leave a partially written record or compressed block in the hourly file. If a partially written compressed block remained and additional compressed blocks were appended, these compressed blocks could not be read by other SiLK tools. If a partially written record remained and additional records were appended, SiLK tools would read the unaligned data as if it were aligned and produce garbage records. Although SiLK 3.6.0 works around the issue on write errors, similar issues can occur if **rwflowappend** is suddenly killed (e.g., by `kill -9`).

When a write error occurs, **rwflowappend** may leave a zero byte file in the data repository. Such files do affect the exit status of **rwfilter(1)**, though **rwfilter** warns about being unable to read the header from the file.

As of SiLK 3.1.0, **rwflowappend** obtains an advisory write lock on the hourly file it is writing, allowing multiple **rwflowappend** processes to write to the same hourly file. File locking may be disabled by using the **--no-file-locking** switch. If this switch is enabled, the administrator must ensure that multiple **rwflowappend** processes do not attempt to write to the same hourly file simultaneously.

rwflowpack

Collect flow data and store it in binary SiLK Flow files

SYNOPSIS

```
rwflowpack [--input-mode=MODE] INPUT_MODE_SPECIFIC_SWITCHES
      [--output-mode=MODE] OUTPUT_MODE_SPECIFIC_SWITCHES
      { --log-destination=DESTINATION
        | --log-pathname=FILE_PATH
        | --log-directory=DIR_PATH [--log-basename=LOG_BASENAME]
          [--log-post-rotate=COMMAND] }
      [--no-file-locking] [--flush-timeout=VAL]
      [--file-cache-size=VAL] [--pack-interfaces]
      [--byte-order=ENDIAN] [--compression-method=COMP_METHOD]
      [--error-directory=DIR_PATH] [--archive-directory=DIR_PATH]
      [--flat-archive] [--post-archive-command=COMMAND]
      [--site-config-file=FILENAME] [--log-level=LEVEL]
      [--log-sysfacility=NUMBER] [--pidfile=FILE_PATH]
      [--no-chdir] [--no-daemon]
```

To collect flow data over the network or directory polling (default):

```
rwflowpack [--input-mode=stream] --sensor-configuration=FILE_PATH
      [--packing-logic=PLUGIN] [--sensor-name=SENSOR]
      [--polling-interval=NUMBER] ...
```

To collect from local files containing flows created by **flowcap(8)**:

```
rwflowpack --input-mode=fcfiles --incoming-directory=DIR_PATH
      --sensor-configuration=FILE_PATH [--packing-logic=PLUGIN]
      [--polling-interval=NUMBER] ...
```

To collect from a single file containing NetFlow v5 PDUs:

```
rwflowpack --input-mode=pdufile --netflow-file=FILE_PATH
      --sensor-configuration=FILE_PATH [--packing-logic=PLUGIN]
      [--sensor-name=SENSOR] ...
```

To respool SiLK Flows without modifying the class, type, or sensor:

```
rwflowpack --input-mode=respool --incoming-directory=DIR_PATH
      [--polling-interval=NUMBER] ...
```

To store the SiLK Flow files on the local machine (default):

```
rwflowpack ... [--output-mode=local-storage]
      --root-directory=DIR_PATH ...
```

To create incremental files to be processed by **rwflowappend(8)**:

```
rwflowpack ... --output-mode=incremental-files
      --incremental-directory=DIR_PATH ...
```

To create incremental files to be processed by **rwflowappend** (deprecated):

```
rwflowpack ... --output-mode=sending --sender-directory=DIR_PATH
      --incremental-directory=DIR_PATH ...
```

Help options:

```
rwflowpack --sensor-configuration=FILE_PATH [--packing-logic=PLUGIN]
      { --verify-sensor-config | --verify-sensor-config=VERBOSE }

rwflowpack --help

rwflowpack --version
```

DESCRIPTION

rwflowpack is a daemon that runs as part of the SiLK flow collection and packing tool-chain. The primary job of **rwflowpack** is to convert each incoming flow record to the SiLK Flow format, categorize each incoming flow record (e.g., as incoming or outgoing), set the sensor value for the record, and determine which hourly file will ultimately store the record.

The settings that **rwflowpack** uses to categorize each flow record are determined by two textual configuration files and compiled code that is referred to as *packing logic*.

The first of the configuration files is **silk.conf(5)** which specifies the classes, types, and sensors to use at your site. There are several different ways to specify the location of this file as detailed in the FILES section below.

The second configuration file is the **sensor.conf(5)** file, whose location is specified via the **--sensor-configuration** switch. This file contains multiple **sensor** blocks, where each block contains information used to categorize flow records captured at that particular sensor. This file also contains **probe** blocks which specify how to collect NetFlow v5, IPFIX, and/or NetFlow v9 flow records, and a mapping of probes to sensors. See the **sensor.conf(5)** manual page for details.

The packing logic uses the information from the **silk.conf** and **sensor.conf** files to *categorize* each flow record. By categorizing a flow record, we mean determine whether the record is incoming or outgoing and assign a class/type pair (also called a *flowtype*) to the record. The flowtype along with the starting hour of the record and the sensor where the record was collected form a triple which determines into which file a flow record is stored. The files that **rwflowpack** produces have the form *flowType*-*sensorName*_YYYYMMDD.HH where *flowType* encodes the class/type pair, *sensorName* is the sensor where the flows were collected, and *YYYYMMDD.HH* is the year, month, day, and hour when the flow started.

For more information on how **rwflowpack** categorizes each flow record and converts data to the SiLK Flow format, see the *SiLK Installation Handbook*, the **sensor.conf(5)** manual page, and the manual page for the packing logic plug-in that **rwflowpack** is using (e.g., **packlogic-twoway(3)** is the default, **packlogic-generic(3)**).

The compiled code for the packing-logic is normally loaded into **rwflowpack** as a run-time. (The administrator may choose to compile the packing logic into **rwflowpack** when building SiLK. See the *SiLK Installation Handbook* for details.) The name of this plug-in is specified in the **packing-logic** statement in the *silk.conf* file. A different location may be specified using the **--packing-logic** switch.

Input Modes

There are several ways to input data to **rwflowpack**, and the method to use is determined by the **--input-mode** switch, with `stream` being the default when the switch is not provided.

`stream`

In `stream` input-mode, **rwflowpack** processes the **probe** statements in the **sensor.conf(5)** file which specify how **rwflowpack** is to capture data from one or more sources. The data is assigned to a sensor based on the probe-sensor mapping in the *sensor.conf* file. **rwflowpack** then categorizes the records, converts them to the SiLK Flow format, and writes them to files.

The sources of data that **rwflowpack** supports are:

- listening for NetFlow v5 packets on a UDP socket
- listening for IPFIX (Internet Protocol Flow Information eXport) packets on a TCP or a UDP socket
- listening for NetFlow v9 packets on a UDP socket
- listening for sFlow v5 packets on a UDP socket
- polling a directory for files containing NetFlow v5 records (see the description of the `pdufile` input-mode for the required format of these files)
- polling a directory for files containing IPFIX records (as generated by **yaf(1)**)
- polling a directory for files containing SiLK Flow records (compare to the `respool` input-mode)

Multiple sources may be specified.

Processing of IPFIX, NetFlow v9, or sFlow is only available when SiLK is compiled with support for libfixbuf-1.7.0 or later. Processing of sFlow records was added in SiLK 3.9.0. libfixbuf is available from http://tools.netsa.cert.org/fixbuf/.

Configuration of `stream` input-mode is specified in the Stream Collection Switches (--input-mode=stream) section below.

`fcfiles`

Instead of having **rwflowpack** capture data itself, you may configure **rwflowpack** to work in conjunction with one or more **flowcap(8)** daemons by specifying `fcfiles` as the input-mode.

In this configuration, each **flowcap** uses the **probe** statements in the **sensor.conf(5)** file to determine how to collect the data. **flowcap** supports the network-based capture methods specified for the `stream` input-mode---i.e., **flowcap** does not support directory polling. **flowcap** writes the data it captures into small files and includes the probe name in each file's header.

Typically, the **flowcap** processes run on separate machines near the router or flow meter that is generating the records. The **rwsender(8)** and **rwreceiver(8)** daemons are often used to move the files from the **flowcap** machines to the **rwflowpack** machine.

rwflowpack polls a directory for the files created by **flowcap**. Once it finds a file, it assigns those records a sensor based on the probe-sensor mapping in the *sensor.conf* file, it categorizes the records, and it writes the records to one or more output files.

Since **rwflowpack** does not capture data in `fcfiles` input-mode, **rwflowpack** does not use the **probe** statements in the *sensor.conf* file, and the statements do not need to be present.

The switches that **rwflowpack** uses in `fcfiles` input-mode are given below (Flowcap Files Collection Switches (--input-mode=fcfiles)).

`pdufile`

Setting the input-mode to `pdufile` tells **rwflowpack** to read a single file containing NetFlow v5 PDU records and then exit. **rwflowpack** does not become a daemon in this input-mode; instead it remains in the foreground, processes the NetFlow file, and exits.

The file must be in the format created by NetFlow Collector: The file's size must be an integer multiple of 1464, where each 1464 byte chunk contains a 24-byte NetFlow v5 header and space for thirty 48-byte NetFlow records. The number of valid records per chunk is specified in the chunk's header. (This is also the format that **rwflowpack** requires in `stream` input-mode when it is polling a directory for NetFlow v5 files.)

To convert single PDU file to a stream of SiLK Flow records, consider using **rwpdu2silk(1)**.

In `pdufile` input-mode, the *sensor.conf* file must define a sensor that maps to a probe that uses the **read-from-file** statement. However, the argument to that statement is ignored, and the argument is typically set to */dev/null*. The NetFlow v5 file's location must be specified with the **--netflow-file** switch. If *sensor.conf* contains multiple sensor blocks, the **--sensor-name** switch is required to tell **rwflowpack** which sensor to use.

See the PDU File Collection Switches (--input-mode=pdufile) section below for additional configuration information.

`respool`

Sometimes it is desirable to pull existing SiLK Flow records from one data repository and use them to create a "mini" data repository (for example, for testing). The `respool` input-mode is one way to accomplish this.

In this configuration, **rwflowpack** polls a directory for SiLK flow files and it uses the *existing* class/type pair and sensor values to determine where to store the flow record. That is, **rwflowpack** puts the data into appropriate hourly file, but it does not change any other settings on the flow records.

To contrast `respool` input-mode with **rwflowpack** processing SiLK Flow files in `stream` input-mode: In `respool` input-mode, the existing class, type, and sensor values are used to determine each record's destination. In the latter, **rwflowpack** treats the records as it would any other newly arrived data, assigning the data to a sensor and re-categorizing the records to assign a class/type pair to them.

Since no categorization occurs in `respool` input-mode, the **--sensor-configuration** and **--packing-logic** switches are not required and not allowed, and their presence causes **rwflowpack** to exit with an error code.

Output Modes

As mentioned above, after **rwflowpack** categorizes a flow record (that is, determines the sensor, class/type, and starting hour for the record), it uses those values to generate the name of the hourly file that will contain that record, and it writes the record to that file.

In order for the record in that file to be retrievable by **rwfilter(1)**, the file must be stored in a SiLK data repository, which is a directory tree of binary SiLK Flow files. The root of this directory tree is called the SILK_DATA_ROOTDIR. The structure of the tree under the root is determined by the **path-format** setting in the **silk.conf(5)** file.

There are two ways to get the files into the SILK_DATA_ROOTDIR; which method is used is determined by **rwflowpack**'s **--output-mode** switch. This switch supports the following values:

`local-storage`

In `local-storage` output-mode, **rwflowpack** creates the hourly SiLK Flow files directly in the data repository, and it writes the records into these files. **rwflowpack** uses `local-storage` output-mode when the **--output-mode** switch is not provided.

`incremental-files`

When the output-mode is `incremental-files`, **rwflowpack** does not create hourly data files directly. Instead, **rwflowpack** creates smaller files (called *incremental files*), and **rwflowpack** relies on the **rwflowappend(8)** daemon to combine the incremental files into hourly files in the final data repository.

In `incremental-files` output-mode, **rwflowpack** uses a single destination directory whose location is specified by the **--incremental-directory** switch. In this directory, **rwflowpack** creates a zero-byte *place holder file* and a corresponding *working file*. The name of the place holder file has a unique, random suffix, and the name of the working file is a dot (`.`) followed by the name of the place holder file.

rwflowpack first writes records to the working files. Periodically (as determined by the value specified to **--flush-timeout**), **rwflowpack** closes all the working files and moves them over the place holder files. Once **rwflowpack** has closed and moved a working file, it no longer takes responsibility for it.

The **rwflowappend** process may poll the incremental-directory itself, or another process may poll that directory and pass the files to **rwflowappend**. If **rwflowpack** and **rwflowappend** are on different machines, an **rwsender(8)**/**rwreceiver(8)** pair may be used to move the files between the machines.

Additional reasons one may want to use `incremental-files` output-mode are to support having multiple data repositories or to allow additional processing of the SiLK Flow records (such as by the Analysis Pipeline (http://tools.netsa.cert.org/analysis-pipeline/)).

Note: This output-mode was introduced in SiLK 3.6.0. This mode is similar to the `sending` output-mode and is meant to replace it. In `incremental-files` output-mode, all writing occurs within the incremental-directory, while the `sending` output-mode uses two directories.

`sending`

This output-mode works similarly to `incremental-files`. The difference is that when **rwflowpack** flushes the open files, it moves the files from the incremental-directory and to the directory specified by the **--sender-directory** switch. Once a file is moved to the sender-directory, **rwflowpack** no longer takes responsibility for it.

As of SiLK 3.6.0, the `sending` output-mode is deprecated.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

Input and Output Mode

As described in the Input Modes section above, **rwflowpack** has multiple ways of getting data, and that data may be stored in one of two methods (c.f., Output Modes). Choosing the modes and configuring each mode are described in the sections below.

--input-mode=*MODE*

Determine how **rwflowpack** gathers data. The default input *MODE* is `stream`. The available input-modes are `stream`, `fcfiles`, `pdufile`, and `respool`.

--output-mode=*MODE*

Determine what **rwflowpack** does with the data as it is packed into SiLK binary files. The default output *MODE* is `local-storage`. The available output-modes are `local-storage`, `incremental-files` and `sending`.

Stream Collection Switches (--input-mode=stream)

In `stream` input-mode, **rwflowpack** uses the **probe** statements in the *sensor.conf* file to capture data, and then **rwflowpack** categorizes the data. The `stream` input-mode is the default when the **--input-mode** switch is not provided. This input-mode accepts the following switches; the **--sensor-configuration** switch is required, and all other switches are optional.

--sensor-configuration=*FILE_PATH*

Give the path to the configuration file that specifies how **rwflowpack** is to capture data and that tells **rwflowpack** whether a record represents an incoming or outgoing flow. The complete syntax of the configuration file is described in the **sensor.conf(5)** manual page; see also the *SiLK Installation Handbook*.

--packing-logic=*PLUGIN*

Specify the plug-in that **rwflowpack** should load, where the plug-in provides functions that determine into which class and type each flow record will be categorized and the format of the files that **rwflowpack** will write. When SiLK has been configured with hard-coded packing logic (i.e., when **--enable-packing-logic** was specified to the **configure** script), this switch will not be present on **rwflowpack**. A default value for this switch may be specified in the **silk.conf(5)** site configuration file (see the description of the **--site-config-file** switch). When *PLUGIN* does not contain a slash (/), **rwflowpack** attempts to find a file named *PLUGIN* in the directories listed in the FILES section. If **rwflowpack** finds the file, it uses that path. If *PLUGIN* contains a slash or if **rwflowpack** does not find the file, **rwflowpack** relies on your operating system's **dlopen(3)** call to find the file. When the SILK_PLUGIN_DEBUG environment variable is non-empty, **rwflowpack** prints status messages to the standard error as it attempts to find and open each of its plug-ins. **rwflowpack** exits if it does not have access to the packing logic functions.

--sensor-name=*SENSOR*

Cause **rwflowpack** to ignore all probes in the sensor configuration file except the probes for *SENSOR*. Only data for *SENSOR* is collected. This allows a common configuration file to be used by multiple **rwflowpack** invocations, yet also allow each **rwflowpack** instance only collect data for a single sensor. There must be a sensor definition for *SENSOR* in the configuration file. When this switch is not present, **rwflowpack** collects and packs data for all sensors.

--polling-interval=*NUMBER*

Specify the number of seconds **rwflowpack** waits between scans of the directories specified by the `poll-directory` settings of the probes in the *sensor.conf* file. The default is 15 seconds.

Flowcap Files Collection Switches (--input-mode=fcfiles)

As described in the Input Modes section above, in `fcfiles` input-mode, **rwflowpack** processes files created by the **flowcap(8)** daemon. In `fcfiles` input-mode, the **--sensor-configuration** and **--incoming-directory** switches are required.

--sensor-configuration=*FILE_PATH*

Give the path to the configuration file that **rwflowpack** consults to determine whether a record represents an incoming or outgoing flow. The complete syntax of the configuration file is described in the **sensor.conf(5)** manual page; see also the *SiLK Installation Handbook*.

--incoming-directory=*DIR_PATH*

Periodically scan the directory *DIR_PATH* for files that have been created by **flowcap**. As **rwflowpack** scans *DIR_PATH*, it ignores a file if its name begins with a dot (`.`) or if its size is 0 bytes. When a file is first detected, **rwflowpack** records its size, and the file must have the same size for two consecutive scans before **rwflowpack** processes it. After the file is successfully processed, **rwflowpack** either moves it to the archive-directory or deletes it. The interval between scans is set by **--polling-interval**. *DIR_PATH* must be a complete directory path.

--packing-logic=*PLUGIN*

Specify the plug-in that **rwflowpack** should load for the packing logic. For more detail, see the description above.

--polling-interval=*NUMBER*

Specify the number of seconds **rwflowpack** waits between polls of the incoming-directory for new files created by **flowcap**. If not given, the default value is 15 seconds.

PDU File Collection Switches (--input-mode=pdufile)

In this input-mode, **rwflowpack** stays in the foreground, processes a single file of NetFlow v5 data, and exits. The **--sensor-configuration** and **--netflow-file** switches are required. The **--sensor-name** switch is also required when the *sensor.conf* file contains more than one sensor. The following switches are available in `pdufile` input-mode:

--sensor-configuration=*FILE_PATH*

Give the path to the configuration file that **rwflowpack** consults to determine whether a record represents an incoming or outgoing flow.

--netflow-file=*FILE_PATH*

Name the full path of the file from which **rwflowpack** reads NetFlow v5 PDUs. This switch is required in PDU File input-mode.

--sensor-name=*SENSOR*

Cause **rwflowpack** to ignore all probes in the sensor configuration file except the probes for *SENSOR*. There must be a sensor definition for *SENSOR* in the configuration file. This switch is required in this input-mode unless the *sensor.conf* file only defines a single sensor.

--packing-logic=*PLUGIN*

Specify the plug-in that **rwflowpack** should load for the packing logic. For more detail, see the description of this switch in the `stream` input-mode.

Respooling Switches (--input-mode=respool)

When the **--input-mode**=`respool` switch is provided, **rwflowpack** polls a directory for SiLK Flow files, and writes the records it finds into new hourly files, leaving the sensor and class/type values unchanged in the records. (See Input Modes above for additional details.) The first of the following switches is required:

--incoming-directory=*DIR_PATH*

Periodically scan the directory *DIR_PATH* for SiLK Flow files to process. As **rwflowpack** scans *DIR_PATH*, it ignores a file if its name begins with a dot (`.`) or if its size is 0 bytes. When a file is first detected, **rwflowpack** records its size, and the file must have the same size for two consecutive scans before **rwflowpack** will process it. After the file is successfully processed, **rwflowpack** either moves it to the archive-directory or deletes it. The interval between scans is set by **--polling-interval**. *DIR_PATH* must be a complete directory path.

--polling-interval=*NUMBER*

Specify the number of seconds **rwflowpack** waits between polls of the incoming-directory. If not given, the default value is 15 seconds.

Local Storage Switches (--output-mode=local-storage)

In `local-storage` output-mode, **rwflowpack** stores SiLK Flow records directly in the data repository. This is the default when the **--output-mode** switch is not provided. When operating in this output-mode, the following switch is required:

--root-directory=*DIR_PATH*

Name the full path of the SILK_DATA_ROOTDIR; that is, the directory under which the files containing the packed SiLK Flow records are stored. **rwflowpack** creates subdirectories below *DIR_PATH* based on the data received and the **path-format** setting in the **silk.conf(5)** file.

Incremental-Files Switches (--output-mode=incremental-files)

As described in the Output Modes section above, the `incremental-files` output-mode tells **rwflowpack** to write incremental-files. The **rwflowappend(8)** daemon combines these incremental-files to create a SiLK data repository. When running in `incremental-files` output-mode, the following switch must be provided:

--incremental-directory=*DIR_PATH*

Name the full path of the destination directory where incremental-files are both created and stored to await action by another process such as **rwflowappend** or **rwsender**. It is recommended that only a single **rwflowpack** process write to this directory.

Sending Mode Switches (--output-mode=sending)

This output-mode is deprecated as of SiLK 3.6.0. This output-mode works similarly to the `incremental-files` output-mode, except **rwflowpack** moves the files to a second directory periodically. Both the following switches are required in `sending` output-mode:

--incremental-directory=*DIR_PATH*

Name the full path of the working directory under which incremental-files are initially created. Periodically (as determined by the **--flush-timeout** switch), **rwflowpack** closes the files in this directory and moves them to the sender-directory. An **rwflowpack** invocation assumes it has full control over the files in this directory. When **rwflowpack** starts, any files in this directory are moved to the sender-directory.

--sender-directory=*DIR_PATH*

Name the full path of the destination directory where incremental-files are moved to await action by another process such as **rwflowappend** or **rwsender**. Once **rwflowpack** moves files to this directory, it no longer takes responsibility for them. The other process (e.g., **rwsender**) is responsible for removing files from this directory. Multiple **rwflowpack** invocations may use a single sender-directory.

General Configuration

The following switches are optional:

--no-file-locking

Do not use advisory write locks. Normally, **rwflowpack** obtains a write lock on an data file prior to writing records to it. The write lock prevents two instances of **rwflowpack** from writing to the same data file simultaneously. However, attempting to use a write lock on some file systems causes **rwflowpack** to exit with an error, and this switch may be used when writing data to these file systems.

--flush-timeout=*VAL*

Set the timeout for flushing any in-memory records to disk to *VAL* seconds. If not specified, the default is 2 minutes (120 seconds). When the output-mode is `local-storage`, this value specifies how often the files are flushed to disk to ensure that any records in memory are written to disk. When using the `incremental-files` or `sending` output-mode, this value specifies how often to close and move the incremental files. See the Output Modes section for details.

--file-cache-size=*VAL*

Set the maximum number of data files to have open for writing at any one time to *VAL*. If not specified, the default is 128 files. The minimum file cache size is 4. This switch also determines how many files **rwflowpack** reads from simultaneously when using probes that poll directories for files (see **sensor.conf(5)**). The maximum number of input files open at any one time is limited to one eighth of *VAL* (with a minimum of 2), and the number of directory polling operations to perform simultaneously is limited to one sixteenth of *VAL* (minimum is 1).

--pack-interfaces

Allow one to override the default file output formats of the packed SiLK Flow files that **rwflowpack** writes. When this switch is present, **rwflowpack** writes additional information into the packed files: the router's SNMP input and output interfaces and the next-hop IP address. (When the *sensor.conf* file contains an `interface-values` attribute whose value is `vlan`, the input and output fields contain the vlan IDs instead of SNMP interface values.) The extra data produced by this switch is useful for determining why traffic is being stored in certain files. Note that this switch only affects newly created files. New records are always appended to an existing file in the file's current output format to maintain file integrity.

--byte-order=*ENDIAN*

Set the byte order for newly created SiLK Flow files. When appending records to an existing file, the byte order of the file is maintained. The argument is one of the following:

`native`
: Use the byte order of the machine where **rwflowpack** is running. This is the default.

`big`
: Use network byte order (big endian) for the flow files.

`little`
: Write the flow files in little endian format.

--compression-method=*COMP_METHOD*

Specify the compression library to use when creating new files. When this switch is not given, newly created files are compressed using the default chosen when SiLK was compiled. When appending records to an existing file, the compression method of the file is maintained. The valid values for *COMP_METHOD* are determined by which external libraries were found when SiLK was compiled. To see the available compression methods and the default method, use the **--help** or **--version** switch. SiLK can support the following *COMP_METHOD* values when the required libraries are available.

none
: Do not compress the output using an external library.

zlib
: Use the **zlib(3)** library for compressing the output. Using zlib produces the smallest output files at the cost of speed.

lzo1x
: Use the *lzo1x* algorithm from the LZO real time compression library for compression. This compression provides good compression with less memory and CPU overhead.

snappy
: Use the *snappy* library for compression, and always compress the output regardless of the destination. This compression provides good compression with less memory and CPU overhead. *Since SiLK 3.13.0.*

best
: Use lzo1x if available, otherwise use snappy if available, otherwise use zlib if available.

--site-config-file=*FILENAME*

Read the SiLK site configuration from the named file *FILENAME*. When this switch is not provided, **rwflowpack** searches for the site configuration file in the locations specified in the FILES section.

Disposal of Input Flow Files

The following switches determine how **rwflowpack** handles input files once it has processed them. These switches have no effect when **rwflowpack** reads all of its data directly from the network. Otherwise, the switches affect the named **--netflow-file** in `pdufile` input-mode, the files read from the **--incoming-directory** in `fcfiles` and `respool` input-mode, and files read from probes that have a `poll-directory` statement (c.f. **sensor.conf(5)**) in `stream` input-mode.

--error-directory=*DIR_PATH*

Move input files that cannot be opened, have an unexpected format, contain an unrecognized probe name in `fcfiles` input-mode, or are not successfully processed into the directory *DIR_PATH*. *DIR_PATH* must be a complete directory path. If this switch is not provided, problem files remain in place and cause **rwflowpack** to exit.

--archive-directory=*DIR_PATH*

Move input files that **rwflowpack** processes successfully into the directory *DIR_PATH*. *DIR_PATH* must be a complete directory path. When this switch is not provided and the input-mode is `pdufile`, the original NetFlow source file is not modified, moved, or deleted. In all other input-modes, no **--archive-directory** switch causes **rwflowpack** to delete each input file after successfully processing it. When the **--flat-archive** switch is also provided, incoming files are moved into the top of *DIR_PATH*; when **--flat-archive** is not given, each file is moved to a subdirectory based on the current UTC time: *DIR_PATH*/*YEAR*/*MONTH*/*DAY*/*HOUR*/. Removing files from the archive-directory is not the job of **rwflowpack**; the system administrator should implement a separate process to clean this directory. This switch is required when the **--post-archive-command** switch is present.

--flat-archive

When archiving input files via the **--archive-directory** switch, move the files into the top of the archive-directory, not into subdirectories of the archive-directory. This switch has no effect if **--archive-directory** is not also specified. This switch may be used to allow another process to watch for new files appearing in the archive-directory.

--post-archive-command=*COMMAND*

Run *COMMAND* on each input file after **rwflowpack** has successfully processed the file and moved the file into the archive-directory. Each occurrence of the string `%s` in *COMMAND* is replaced with the full path to the input file in the archive-directory, and each occurrence of `%%` is replaced with `%`. If any other character follows `%`, **rwflowpack** exits with an error. When using this feature, the **--archive-directory** switch must be specified. See also the **rwpollexec(8)** daemon.

Logging and Daemon Configuration

One of the following mutually-exclusive switches is required:

--log-destination=*DESTINATION*

Specify the destination where logging messages are written. When *DESTINATION* begins with a slash `/`, it is treated as a file system path and all log messages are written to that file; there is no log rotation. When *DESTINATION* does not begin with `/`, it must be one of the following strings:

`none`
: Messages are not written anywhere.

`stdout`
: Messages are written to the standard output.

`stderr`
: Messages are written to the standard error.

`syslog`
: Messages are written using the **syslog(3)** facility.

`both`
: Messages are written to the syslog facility and to the standard error (this option is not available on all platforms).

--log-directory=*DIR_PATH*

Use *DIR_PATH* as the directory where the log files are written. *DIR_PATH* must be a complete directory path. The log files have the form

```
DIR_PATH/LOG_BASENAME-YYYYMMDD.log
```

where *YYYYMMDD* is the current date and *LOG_BASENAME* is the application name or the value passed to the **--log-basename** switch when provided. The log files are rotated: At midnight local time, a new log is opened, the previous file is closed, and the command specified by **--log-post-rotate** is invoked on the previous day's log file. (Old log files are not removed by **rwflowpack**; the administrator should use another tool to remove them.) When this switch is provided, a process-ID file (PID) is also written in this directory unless the **--pidfile** switch is provided.

--log-pathname=*FILE_PATH*

Use *FILE_PATH* as the complete path to the log file. The log file is not rotated.

The following set of switches is optional:

--log-level=*LEVEL*

Set the severity of messages that are logged. The levels from most severe to least are: `emerg`, `alert`, `crit`, `err`, `warning`, `notice`, `info`, `debug`. The default is `info`.

--log-sysfacility=*NUMBER*

Set the facility that **syslog(3)** uses for logging messages. This switch takes a number as an argument. The default is a value that corresponds to `LOG_USER` on the system where **rwflowpack** is running. This switch produces an error unless **--log-destination**=syslog is specified.

--log-basename=*LOG_BASENAME*

Use *LOG_BASENAME* in place of the application name in the name of *log* files in the log directory. See the description of the **--log-directory** switch. This switch does **not** affect the name of the process-ID file.

--log-post-rotate=*COMMAND*

Run *COMMAND* on the previous day's log file after log rotation. When this switch is not specified, the previous day's log file is compressed with **gzip(1)**. When the switch is specified and *COMMAND* is the empty string, no action is taken on the log file. Each occurrence of the string `%s` in *COMMAND* is replaced with the full path to the log file, and each occurrence of `%%` is replaced with `%`. If any other character follows `%`, **rwflowpack** exits with an error. Specifying this switch without also using **--log-directory** is an error.

--pidfile=*FILE_PATH*

Set the complete path to the file in which **rwflowpack** writes its process ID (PID) when it is running as a daemon. No PID file is written when **--no-daemon** is given. When this switch is not present, no PID file is written unless the **--log-directory** switch is specified, in which case the PID is written to *LOGPATH/rwflowpack.pid*.

--no-chdir

Do not change directory to the root directory. When **rwflowpack** becomes a daemon process, it changes its current directory to the root directory so as to avoid potentially running on a mounted file system. Specifying **--no-chdir** prevents this behavior, which may be useful during debugging. The application does not change its directory when **--no-daemon** is given.

--no-daemon

Force **rwflowpack** to run in the foreground---it does not become a daemon process. This may be useful during debugging.

Help Options

--verify-sensor-config

--verify-sensor-config=*VERBOSE*

Verify that the syntax of the sensor configuration file is correct and then exit **rwflowpack**. If the file is incorrect or if it does not define any sensors, an error message is printed and **rwflowpack** exits abnormally. If the file is correct and no argument is provided to the **--verify-sensor-config** switch, **rwflowpack** simply exits with status 0. If an argument (other than the empty string and `0`) is provided to the switch, the names of the probes and sensors found in the sensor configuration file are printed to the standard output, and then **rwflowpack** exits.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

ENVIRONMENT

SILK_IPFIX_PRINT_TEMPLATES

When set to 1, **rwflowpack** writes messages to the log file describing each IPFIX and NetFlow v9 template it receives. This is equivalent to adding `show-templates` to the `log-flags` setting for each probe in the *sensor.conf* file. See the **sensor.conf(5)** manual page for the format of these messages. *Since SiLK 3.8.2.*

SILK_LIBFIXBUF_SUPPRESS_WARNINGS

When set to 1, **rwflowpack** disables all warning messages generated by libfixbuf. These warning messages include out-of-sequence packets, data records not having a corresponding template, record count discrepancies, and issues decoding list elements. *Since SiLK 3.10.0.*

SILK_CONFIG_FILE

This environment variable is used as the value for the **--site-config-file** when that switch is not provided.

SILK_DATA_ROOTDIR

This environment variable specifies the root directory of data repository. When the output-mode is `sending`, **rwflowpack** may use this environment variable when searching for the SiLK site configuration file. See the FILES section for details.

SILK_PATH

This environment variable gives the root of the install tree. When searching for configuration files, **rwflowpack** may use this environment variable. See the FILES section for details.

SILK_PLUGIN_DEBUG

When set to 1, **rwflowpack** print status messages to the standard error as it tries to open the packing logic plug-in.

FILES

sensor.conf

The location of this file must be specified by the **--sensor-configuration** switch. This file specifies **probe** blocks that tell **rwflowpack** how to capture data when the **--input-mode** is `stream`. The file also contains **sensor** blocks that map probes to sensors and that the packing-logic uses to determine the category of each flow record. The syntax of this file is described in the **sensor.conf(5)** manual page.

${SILK_CONFIG_FILE}

ROOT_DIRECTORY/silk.conf

${SILK_PATH}/share/silk/silk.conf

${SILK_PATH}/share/silk.conf

/usr/local/share/silk/silk.conf

/usr/local/share/silk.conf

Possible locations for the SiLK site configuration file which are checked when the **--site-config-file** switch is not provided. When **rwflowpack** is running in `local-storage` output-mode, *ROOT_DIRECTORY/* is the directory specified to the **--root-directory** switch. When the output-mode is `sending`, *ROOT_DIRECTORY/* is either the value specified in the SILK_DATA_ROOTDIR environment variable or the default data repository directory compiled into **rwflowpack** (/data).

${SILK_PATH}/lib64/silk/

${SILK_PATH}/lib64/

${SILK_PATH}/lib/silk/

${SILK_PATH}/lib/

/usr/local/lib64/silk/

/usr/local/lib64/

/usr/local/lib/silk/

/usr/local/lib/

Directories that **rwflowpack** checks when attempting to load the packing-logic plug-in.

SEE ALSO

SiLK Installation Handbook, **sensor.conf(5)**, **silk.conf(5)**, **packlogic-twoway(3)**, **packlogic-generic(3)**, **flowcap(8)**, **rwfilter(1)**, **rwflowappend(8)**, **rwreceiver(8)**, **rwsender(8)**, **rwpollexec(8)**, **rwpdu2silk(1)**, **rwpackchecker(8)**, **silk(7)**, **gzip(1)**, **yaf(1)**, **dlopen(3)**, **zlib(3)**, **syslog(3)**

NOTES

As SiLK 3.6.0, the `incremental-files` output-mode should be used in place of the `sending` output-mode that existed in prior releases of **rwflowpack**. See Output Modes for details.

For administrators that use the `sending` output-mode in SiLK 3.5 or older and upgrade to SiLK 3.6 or later: Any incremental files that the older version of **rwflowpack** leaves in the incremental-directory will **not** be moved to the sender-directory by the new version of **rwflowpack**. Those files will need to be moved by hand.

rwflowpack does not check the integrity of the data file before appending records to it.

When the disk becomes full or other write errors occur, **rwflowpack** may leave partially written records or partially written compressed blocks in the files it has open. For each file where a partially written compressed block remains and additional compressed blocks are appended, the newly appended compressed blocks are not readable by other SiLK tools. For each file where a partially written record remains and additional records are appended, other SiLK tools will read the unaligned data as if it were aligned and produce garbage records. Partially writes may also occur if **rwflowpack** is suddenly killed (e.g., by `kill -9`).

When a write error occurs, **rwflowpack** may leave a zero byte file in the data repository. Such files do affect the exit status of **rwfilter(1)**, though **rwfilter** warns about being unable to read the header from the file.

rwflowpack obtains an advisory write lock on the hourly file it is writing, allowing multiple **rwflowpack** processes to write to the same hourly file. File locking may be disabled by using the **--no-file-locking** switch. If this switch is enabled, the administrator must ensure that multiple **rwflowpack** processes do not attempt to write to the same hourly file simultaneously.

rwguess

Determine which SNMP interfaces are active

SYNOPSIS

```
rwguess [{ --top=NUM | --print-all }] PDU_FILE [PDU_FILE...]

rwguess --help

rwguess --version
```

DESCRIPTION

rwguess is deprecated as of SiLK 3.8.3 and it will be removed in the SiLK 4.0 release. Replace invocations of **rwguess** with **rwpdu2silk(1)** and either **rwstats(1)** or **rwuniq(1)** as shown in EXAMPLES.

rwguess reads NetFlow v5 PDUs from file(s) specified on the command line and counts the number of flow records that are seen on each input and output SNMP interface. Once all input has been processed, **rwguess** sorts the SNMP interfaces by the number of records each interface saw, and prints the two sorted lists, one for the input interfaces and one for the output interfaces. By default, only the top-10 interfaces are printed; the number of rows printed may be changed with the **--top** switch.

When the **--print-all** switch is specified, the results are printed in SNMP interface order, with one column for the input record count and another for the output record count, and one row for each interface that saw traffic.

The purpose of **rwguess** is to help one configure the `sensor` blocks in the **silk.conf(5)** file used by **rwflowpack(8)** to categorize flow records into classes and types.

The PDU files are expected to be in the form created by NetFlow Collector: Each file's size must be an integer multiple of 1464, where each 1464 byte chunk contains a 24 byte NetFlow v5 header and space for thirty 48 byte NetFlow records. The number of valid records per chunk is specified in the PDU header.

To convert a PDU file to a stream of SiLK Flow records, use **rwpdu2silk(1)**.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--top=*NUM*

Print the top *NUM* interfaces for each of input and output. If not specified, the default is to print the top 10 interfaces.

--print-all

Print all SNMP interfaces that saw records, sorted by the SNMP interface number. This switch disables top-N printing.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

EXAMPLES

rwguess is deprecated. This section demonstrates how to get equivalent functionality by piping the output from **rwpdu2silk(1)** into either **rwstats(1)** or **rwuniq(1)**.

In the following examples, the dollar sign (**$**) represents the shell prompt. The text after the dollar sign represents the command line. Lines have been wrapped for improved readability, and the back slash (\) is used to indicate a wrapped line.

Top-N List

By default, **rwguess** creates a top-10 list of SNMP interfaces.

```
$ rwguess file.pdu
Top 10 (of 36) SNMP Input Interfaces
Index|  Input_Recs|
   54|        3466|
   38|        1374|
   84|         770|
   88|         746|
   56|         737|
   68|         513|
  106|         508|
   62|         373|
  114|         323|
    8|         321|

Top 10 (of 37) SNMP Output Interfaces
Index| Output_Recs|
   54|        3507|
   38|         885|
   98|         699|
   84|         673|
   88|         671|
   56|         605|
   58|         538|
  106|         501|
   92|         460|
   62|         380|
```

Use **rwpdu2silk** to convert the file to SiLK flow format, and pipe the result to **rwstats**. You must invoke **rwstats** twice, once the input interface (**--field=in**) and once for the output interface (**--field=out**). The **--copy-input** switch allows the second **rwstats** command to read output from **rwpdu2silk**.

```
$ rwpdu2silk file.pdu  \
  | rwstats --count=10 --fields=in --copy-input=- --output-path=stderr \
  | rwstats --count=10 --fields=out
INPUT: 12056 Records for 36 Bins and 12056 Total Records
OUTPUT: Top 10 Bins by Records
   in|   Records|  %Records|   cumul_%|
   54|      3466| 28.750663| 28.750663|
   38|      1374| 11.398869| 40.149532|
   84|       770|  6.388336| 46.537868|
   88|       746|  6.193106| 52.730975|
   56|       737|  6.117718| 58.848693|
   68|       513|  4.261379| 63.110072|
  106|       508|  4.216760| 67.326831|
   62|       373|  3.094729| 70.421560|
  114|       323|  2.681877| 73.103437|
    8|       321|  2.666285| 75.769722|
INPUT: 12056 Records for 37 Bins and 12056 Total Records
OUTPUT: Top 10 Bins by Records
  out|   Records|  %Records|   cumul_%|
   54|      3507| 29.089205| 29.089205|
   38|       885|  7.347980| 36.437185|
   98|       699|  5.801735| 42.238920|
   84|       673|  5.588923| 47.827843|
   88|       671|  5.572502| 53.400345|
   56|       605|  5.022807| 58.423152|
   58|       538|  4.462497| 62.885649|
  106|       501|  4.155802| 67.041451|
   92|       460|  3.821822| 70.863273|
   62|       380|  3.157428| 74.020701|
```

Seeing all interfaces

The **--print-all** switch shows all interfaces.

```
$ rwguess --print-all file2.pdu
Index|  Input_Recs| Output_Recs|
   10|       17099|       17115|
  172|        7893|        7893|
  192|       25008|       24992|
```

Use **rwuniq** to generate similar output, though you must run **rwuniq** twice (as with **rwstats** in the previous example).

```
$ rwpdu2silk file2.pdu   \
  | rwuniq --sort --fields=in --copy-input=- --output-path=stderr  \
  | rwuniq --sort --fields=out
   in|   Records|
   10|     17099|
  172|      7893|
  192|     25008|
  out|   Records|
```

```
  10|     17115|
 172|      7893|
 192|     24992|
```

SEE ALSO

rwpdu2silk(1), **rwstats(1)**, **rwuniq(1)**, **rwflowpack(8)**, **silk.conf(5)**, **silk(7)**

rwpackchecker

Find unusual patterns that may indicate a corrupt file

SYNOPSIS

```
rwpackchecker [--value=TEST=VALUE] [--allowable-count=TEST=ALLOWED]
      [--print-all]
      {[--xargs] | [--xargs=FILENAME] | [FILE [FILE ...]]}

rwpackchecker --help

rwpackchecker --version
```

DESCRIPTION

rwpackchecker reads SiLK Flow records and checks for `unusual` patterns that *may* indicate the file has been corrupted.

rwpackchecker has a default series of tests it runs on every flow record in an input file. Each default test has two numbers associated with it: a value threshold and an allowed count threshold. A test compares a value on the flow record to the value threshold, and if the value violates the threshold, a counter for that test is incremented. In addition, if the flow record violates the value threshold for any test, a global counter is incremented to denote a suspect record.

Once **rwpackchecker** finishes processing a file, it determines whether the file appears to be valid. A file is considered valid if either

- the global counter of suspect records is 0, or
- no test has a counter that exceeds the test's allowed count threshold

If **rwpackchecker** determines that all files are valid, it does not print any output by default. If **rwpackchecker** does find an invalid file, it will print the name of the input file, the global number of suspect records it found, and information for those tests where the counter exceeds the allowed count threshold.

As an example, if there are 10 tests and the count threshold for each is 10, it is possible for the global suspect counter to be 90 and for **rwpackchecker** to consider the file valid.

To force **rwpackchecker** to print the results for all tests and for all input files, specify the **--print-all** switch.

Some of the tests that run by default include checking the number of packets, the bytes per second ratio, the bytes per packet ratio, and the bytes per packet ratio for a particular protocol (TCP, UDP, and ICMP).

The **--value** and **--allowable-count** switches modify the value threshold and allowed count threshold for a test, respectively. The argument to the switch is the test name and the threshold, separated by a equals sign (`=`). Repeat the switches to set multiple thresholds. For example, to change the value thresholds for the max-bytes test to 20000 and for the max-packets test to 15000, specify the following:

```
rwpackchecker --value=max-bytes=20000 --value max-packets=15000 ...
```

To get the most value from **rwpackchecker**, one should customize it for the particular site where it is being used, since the default value for a threshold may or may not be unusual for a particular installation. For example, a router that has Ethernet connections should have no more than 1500 bytes per packet, since that is the Ethernet MTU; however, the default value for that ratio is 16384. In some cases the default value is the largest value that a SiLK IPv4 hourly repository file can store, making it impossible for a record to violate the threshold.

rwpackchecker supports additional tests which are not run by default. Representative tests include checking whether an IP is (not) in an IPset or whether a port is (not) in an integer list. To run an additional test, specify the name of the test using the **--value** switch and provide an argument for the test.

To see all of the tests that **rwpackchecker** supports as well as the value threshold and allowed count threshold for each test, see the OPTIONS section below, or run **rwpackchecker** with the **--help** switch.

rwpackchecker reads SiLK Flow records from the files named on the command line or from the standard input when no file names are specified and **--xargs** is not present. To read the standard input in addition to the named files, use `-` or `stdin` as a file name. If an input file name ends in `.gz`, the file is uncompressed as it is read. When the **--xargs** switch is provided, **rwpackchecker** reads the names of the files to process from the named text file or from the standard input if no file name argument is provided to the switch. The input to **--xargs** must contain one file name per line.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--value=*TEST*=*VALUE*

Set the value of *TEST* to the specified *VALUE*; separate the test name from value by `=`. The available *TEST*s are given below; the test name can be shortened to the shortest unique prefix. The form of *VALUE* depends on the type of *TEST*:

- If *TEST* expects a minimum or maximum, *VALUE* should be a number.
- If *TEST* expects a list of IPs, *VALUE* should the name of a file containing an IPset (see **rwsetbuild(1)**).
- If *TEST* expects a list of numbers (for example, ports or protocols), *VALUE* should contain a comma separated list of integers and integer-ranges where a range is two integers separated by a hyphen (`-`).

Repeat this switch for each value that you wish to set.

--allowable-count=*TEST*=*ALLOWED*

Allow the named *TEST* to be violated *ALLOWED* of times before treating it as `unusual`. *ALLOWED* is an integer value. Separate the test name from the allowed count by `=`. Repeat this switch for each allowable count you wish to set.

--print-all

Print the result of all tests for all input files. Normally only tests that are deemed `unusual` are printed.

--xargs

--xargs=*FILENAME*

Read the names of the input files from *FILENAME* or from the standard input if *FILENAME* is not provided. The input is expected to have one filename per line. **rwpackchecker** opens each named file in turn and reads records from it as if the filenames had been listed on the command line.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

The following tests are always run:

min-bpp-ratio=*NUMBER*

Byte-per-packet ratio is less than *NUMBER*. Default value: 1. Allowed count: 0.

max-bpp-ratio=*NUMBER*

Byte-per-packet ratio is greater than *NUMBER*. Default value: 16384. Allowed count: 0.

min-bps-ratio=*NUMBER*

Byte-per-second ratio is less than *NUMBER*. Default value: 0. Allowed count: 0.

max-bps-ratio=*NUMBER*

Byte-per-second ratio is greater than *NUMBER*. Default value: 4294967295. Allowed count: 0.

min-packets=*NUMBER*

Packet count is less than *NUMBER*. Default value: 1. Allowed count: 0.

max-packets=*NUMBER*

Packet count is greater than *NUMBER*. Default value: 67108864. Allowed count: 0.

min-bytes=*NUMBER*

Byte count is less than *NUMBER*. Default value: 1. Allowed count: 0.

max-bytes=*NUMBER*

Byte count is greater than *NUMBER*. Default value: 4294967295. Allowed count: 0.

min-tcp-bpp-ratio=*NUMBER*

TCP byte-per-packet ratio is less than *NUMBER*. Default value: 1. Allowed count: 0.

max-tcp-bpp-ratio=*NUMBER*

TCP byte-per-packet ratio is greater than *NUMBER*. Default value: 16384. Allowed count: 0.

min-udp-bpp-ratio=*NUMBER*

UDP byte-per-packet ratio is less than *NUMBER*. Default value: 1. Allowed count: 0.

max-udp-bpp-ratio=*NUMBER*

UDP byte-per-packet ratio is greater than *NUMBER*. Default value: 16384. Allowed count: 0.

min-icmp-bpp-ratio=*NUMBER*

ICMP byte-per-packet ratio is less than *NUMBER*. Default value: 1. Allowed count: 0.

max-icmp-bpp-ratio=*NUMBER*

ICMP byte-per-packet ratio is greater than *NUMBER*. Default value: 16384. Allowed count: 0.

The following tests are only run when the **--value** switch is used to specify a value for the test.

match-protocol=*LIST*

Protocol is present in *LIST*. No default. Allowed count: 0.

nomatch-protocol=*LIST*

Protocol is not present in *LIST*. No default. Allowed count: 0.

match-flags=*LIST*

TCP Flag Combination is present in *LIST*. No default. Allowed count: 0.

nomatch-flags=*LIST*

TCP Flag Combination is not present in *LIST*. No default. Allowed count: 0.

match-sip=*IPSET_FILE*

Source IP is present in *IPSET_FILE*. No default. Allowed count: 0.

nomatch-sip=*IPSET_FILE*

Source IP is not present in *IPSET_FILE*. No default. Allowed count: 0.

match-dip=*IPSET_FILE*

Destination IP is present in *IPSET_FILE*. No default. Allowed count: 0.

nomatch-dip=*IPSET_FILE*

Destination IP is not present in *IPSET_FILE*. No default. Allowed count: 0.

match-sport=*LIST*

Source Port is present in *LIST*. No default. Allowed count: 0.

nomatch-sport=*LIST*

Source Port is not present in *LIST*. No default. Allowed count: 0.

match-dport=*LIST*

Destination Port is present in *LIST*. No default. Allowed count: 0.

nomatch-dport=*LIST*

Destination Port is not present in *LIST*. No default. Allowed count: 0.

match-nhip=*IPSET_FILE*

Next Hop IP is present in *IPSET_FILE*. No default. Allowed count: 0.

nomatch-nhip=*IPSET_FILE*

Next Hop IP is not present in *IPSET_FILE*. No default. Allowed count: 0.

match-input=*LIST*

SNMP Input is present in *LIST*. No default. Allowed count: 0.

nomatch-input=*LIST*

SNMP Input is not present in *LIST*. No default. Allowed count: 0.

match-output=*LIST*

SNMP Output is present in *LIST*. No default. Allowed count: 0.

nomatch-output=*LIST*

SNMP Output is not present in *LIST*. No default. Allowed count: 0.

EXAMPLES

In these examples, the dollar sign (`$`) represents the shell prompt and a backslash (`\`) is used to continue a line for better readability. The examples do not use the optional `=` between the **--value** switch and the switch's argument for better readability.

Given the SiLK Flow file *data.rw* where the number of flows with various byte and packet counts are described by this table:

```
Number of flows    bytes <= 2000000    bytes > 2000000     TOTAL
packets <= 500          379303                308          379611
packets >  500          119586               2679          122265
TOTAL                   498889               2987          501876
```

Running **rwpackchecker**:

```
$ rwpackchecker --value max-bytes=2000000           \
       --value max-packets=500  data.rw
data.rw:
             122573/501876 flows are bad or unusual
             122265 flows where Packet Count > 500
               2987 flows where Byte Count > 2000000
```

The counts for the individual tests are greater than the overall total since 2679 records fall into both categories.

To see the effect of the **--allowable-count** switch:

```
$ rwpackchecker --value max-packets=500                             \
       --value max-bytes=2000000 --allowable max-bytes=3000 data.rw
data.rw:
             122573/501876 flows are bad or unusual
             122265 flows where Packet Count > 500

$ rwpackchecker --value max-bytes=2000000                           \
       --value max-packets=500 --allowable max-packets=150000 data.rw
data.rw:
             122573/501876 flows are bad or unusual
               2987 flows where Byte Count > 2000000
```

In each case the total number of unusual flows did not change; the violation of the other limit is still noted, even when the test is not printed since the test's allowed count threshold was not reached.

When the allowed count thresholds for none of the tests are reached, **rwpackchecker** produces no output by default:

```
$ rwpackchecker --value max-bytes=2000000 --allowable max-bytes=3000 \
       --value max-packets=500 --allowable max-packets=150000 data.rw
$
```

Specify the **--print-all** switch to print the results:

```
$ rwpackchecker --value max-bytes=2000000 --allowable max-bytes=3000 \
       --value max-packets=500 --allowable max-packets=15000         \
       --print-all data.rw
data.rw:
             122573/501876 flows are bad or unusual
                  0 flows where BPP Calculation is incorrect
                  0 flows where Elapsed Time > 4096
                  0 flows where Byte/Packet Ratio < 1
                  0 flows where Byte/Packet Ratio > 16384
                  0 flows where Byte/Second Ratio < 0
                  0 flows where Byte/Second Ratio > 4294967295
                  0 flows where Packet Count < 1
             122265 flows where Packet Count > 500
                  0 flows where Byte Count < 1
               2987 flows where Byte Count > 2000000
                  0 flows where TCP Byte/Packet Ratio < 1
                  0 flows where TCP Byte/Packet Ratio > 16384
                  0 flows where UDP Byte/Packet Ratio < 1
                  0 flows where UDP Byte/Packet Ratio > 16384
                  0 flows where ICMP Byte/Packet Ratio < 1
                  0 flows where ICMP Byte/Packet Ratio > 16384
```

SEE ALSO

rwflowpack(8), **rwsetbuild(1)**, **silk(7)**

rwpollexec

Monitor a directory for files and execute a command on them

SYNOPSIS

```
rwpollexec --incoming-directory=DIR_PATH --command=COMMAND
      --error-directory=DIR_PATH [--archive-directory=DIR_PATH]
      [--flat-archive] [--simultaneous=NUM]
      [--timeout=SIGNAL,SECS [--timeout=SIGNAL,SECS ...]]
      [--polling-interval=NUM]
      { --log-destination=DESTINATION
        | --log-pathname=FILE_PATH
        | --log-directory=DIR_PATH [--log-basename=LOG_BASENAME]
          [--log-post-rotate=COMMAND] }
      [--log-level=LEVEL] [--log-sysfacility=NUMBER]
      [--pidfile=FILE_PATH] [--no-chdir] [--no-daemon]

rwpollexec --help

rwpollexec --version
```

DESCRIPTION

rwpollexec is a daemon that monitors a directory for incoming files and executes a given command on each file. If the command runs successfully on a file, the file is either moved to an archive directory or deleted. If the command runs unsuccessfully or is terminated by a signal, the file is moved to an error directory.

rwpollexec executes a single command on each file. If you need to run multiple commands on a file, create a script to run the commands and have **rwpollexec** execute the script.

The **--simultaneous** switch specifies the maximum number of invocations of *COMMAND* that **rwpollexec** will run concurrently. The default is one, which causes **rwpollexec** to process the files one at a time.

If there is a possibility that the command will "hang" and cause **rwpollexec** to stop processing files, you may wish to specify that **rwpollexec** send a signal to the command after it has been running for some number of seconds by using the **--timeout** switch. This switch may be repeated to send different signals after various times.

When **rwpollexec** is signaled to exit, it waits for all running commands to terminate before exiting. If a command has "hung", **rwpollexec** does not exit until that command is killed, or until **rwpollexec** itself is sent a SIGKILL.

As **rwpollexec** scans the incoming file directory, it ignores a file if its size is 0 bytes or if its name begins with a dot (`.`). On each scan, if **rwpollexec** detects a file name that was not present in the previous scan, it records the name and size of the file. If the file has a different size on the next scan, the new size is recorded. Once the file has the same size on two consecutive scans, **rwpollexec** executes the command on the file.

If the exit status of running the command of a file is zero, **rwpollexec** deletes the file unless the **--archive-directory** switch is specified, in which case the file is moved to that directory or to a subdirectory of that directory depending on whether the **--flat-archive** switch is specified.

If the exit status of the command is non-zero, the file is moved to the error directory. **rwpollexec** does not provide a method to re-try a command that fails.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

--incoming-directory=*DIR_PATH*

Periodically scan the directory *DIR_PATH* for files on which to run the command specified by the **--command** switch. As **rwpollexec** scans *DIR_PATH*, it ignores a file if its name begins with a dot (`.`) or if its size is 0 bytes. When a file is first detected, its size is recorded, and the file must have the same size for two consecutive scans before **rwpollexec** will execute the command on it. The interval between scans is set by **--polling-interval**. *DIR_PATH* must be a complete directory path. This switch is required.

--command=*COMMAND*

Run *COMMAND* on each file noticed in the directory specified by **--incoming-directory**. Each occurrence of the string `%s` in *COMMAND* is replaced with the full path to the file, and each occurrence of `%%` is replaced with `%`. If any other character follows `%`, **rwpollexec** exits with an error. If the exit status of *COMMAND* is zero, **rwpollexec** deletes the file unless the **--archive-directory** switch is specified, in which case **rwpollexec** moves the file to that directory. If the command exits with a non-zero status or is terminated by a signal, **rwpollexec** moves the file to the directory specified by **--error-directory**. This switch is required.

COMMAND is interpreted by the shell used by **rwpollexec**. When the `SILK_RWPOLLEXEC_SHELL` environment variable is set, its value is used as the shell. Otherwise, **rwpollexec** determines the shell as described in the FILES section. Any output on `stdout` or `stderr` from *COMMAND* will appear in the log when the log messages are being written to a local log file.

--error-directory=*DIR_PATH*

Move to this directory files where *COMMAND* either runs unsuccessfully (i.e., has a non-zero exit status) or terminates by a signal. *DIR_PATH* must be a complete directory path. This switch is required.

--archive-directory=*DIR_PATH*

Move to this directory the files where *COMMAND* runs successfully (i.e., has an exit status of zero). *DIR_PATH* must be a complete directory path. If this switch is not supplied, **rwpollexec** will delete these files instead. When the **--flat-archive** switch is also provided, incoming files are moved into *DIR_PATH*; when **--flat-archive** is not given, each file is moved to a subdirectory of *DIR_PATH* based on the current local time: *DIR_PATH*/*YEAR*/*MONTH*/*DAY*/*HOUR*/. Removing files from the archive-directory is not the job of **rwpollexec**; the system administrator should implement a separate process to clean this directory.

--flat-archive

When archiving input files via the **--archive-directory** switch, move the files into the top of the archive-directory, not into subdirectories of the archive-directory. This switch has no effect if **--archive-directory** is not also specified. This switch may be used to allow another process to watch for new files appearing in the archive-directory.

--simultaneous=*NUM*

Allow a maximum of *NUM* commands to be executed simultaneously. The default is one, which allows only one command to be run at a time. The maximum value allowed for this switch is 50.

--timeout=*SIGNAL,SECS*

Send *SIGNAL* to the running command if it has been executing for *SECS* seconds. *SIGNAL* may be a signal name, with or without a `SIG` prefix, or a signal number. A list of signals can be determined by running the command `kill -l` at a shell prompt (cf. **kill(1)**). This switch may be repeated to send different signals after various amounts of time.

--polling-interval=*NUM*

Configure **rwpollexec** to check the incoming directory for new files every *NUM* seconds. The default polling interval is 15 seconds.

One of the following logging switches is required:

--log-destination=*DESTINATION*

Specify the destination where logging messages are written. When *DESTINATION* begins with a slash `/`, it is treated as a file system path and all log messages are written to that file; there is no log rotation. When *DESTINATION* does not begin with `/`, it must be one of the following strings:

`none`
: Messages are not written anywhere.

`stdout`
: Messages are written to the standard output.

`stderr`
: Messages are written to the standard error.

`syslog`
: Messages are written using the **syslog(3)** facility.

`both`
: Messages are written to the syslog facility and to the standard error (this option is not available on all platforms).

--log-directory=*DIR_PATH*

Use *DIR_PATH* as the directory where the log files are written. *DIR_PATH* must be a complete directory path. The log files have the form

```
DIR_PATH/LOG_BASENAME-YYYYMMDD.log
```

where *YYYYMMDD* is the current date and *LOG_BASENAME* is the application name or the value passed to the **--log-basename** switch when provided. The log files are rotated: At midnight local time, a new log is opened, the previous file is closed, and the command specified by **--log-post-rotate** is invoked on the previous day's log file. (Old log files are not removed by **rwpollexec**; the administrator should use another tool to remove them.) When this switch is provided, a process-ID file (PID) is also written in this directory unless the **--pidfile** switch is provided.

--log-pathname=*FILE_PATH*

Use *FILE_PATH* as the complete path to the log file. The log file is not rotated.

The following set of switches is optional:

--log-level=*LEVEL*

Set the severity of messages that are logged. The levels from most severe to least are: `emerg`, `alert`, `crit`, `err`, `warning`, `notice`, `info`, `debug`. The default is `info`.

--log-sysfacility=*NUMBER*

Set the facility that **syslog(3)** uses for logging messages. This switch takes a number as an argument. The default is a value that corresponds to `LOG_USER` on the system where **rwpollexec** is running. This switch produces an error unless **--log-destination**=syslog is specified.

--log-basename=*LOG_BASENAME*

Use *LOG_BASENAME* in place of the application name in the name of *log* files in the log directory. See the description of the **--log-directory** switch. This switch does **not** affect the name of the process-ID file.

--log-post-rotate=*COMMAND*

Run *COMMAND* on the previous day's log file after log rotation. When this switch is not specified, the previous day's log file is compressed with **gzip(1)**. When the switch is specified and *COMMAND* is the empty string, no action is taken on the log file. Each occurrence of the string `%s` in *COMMAND* is replaced with the full path to the log file, and each occurrence of `%%` is replaced with `%`. If any other character follows `%`, **rwpollexec** exits with an error. Specifying this switch without also using **--log-directory** is an error.

--pidfile=*FILE_PATH*

Set the complete path to the file in which **rwpollexec** writes its process ID (PID) when it is running as a daemon. No PID file is written when **--no-daemon** is given. When this switch is not present, no PID file is written unless the **--log-directory** switch is specified, in which case the PID is written to *LOGPATH/rwpollexec.pid*.

--no-chdir

Do not change directory to the root directory. When **rwpollexec** becomes a daemon process, it changes its current directory to the root directory so as to avoid potentially running on a mounted file system. Specifying **--no-chdir** prevents this behavior, which may be useful during debugging. The application does not change its directory when **--no-daemon** is given.

--no-daemon

Force **rwpollexec** to run in the foreground---it does not become a daemon process. This may be useful during debugging.

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

ENVIRONMENT

SILK_RWPOLLEXEC_SHELL

The shell to use for executing commands. If this variable is not set, **rwpollexec** tests the list of shells specified in FILES to find a shell that uses a member of the **execl(3)** family of functions to run the command. More details are available in the BUGS section.

FILES

${SILK_RWPOLLEXEC_SHELL}

/bin/sh

/bin/bash

/bin/ksh

/usr/bin/sh

/usr/bin/bash

/usr/bin/ksh

Shells that **rwpollexec** may use to invoke the command specified by **--command**. The shell specified in SILK_RWPOLLEXEC_SHELL is always used when that variable is set. Otherwise, **rwpollexec** checks the list of shells to find one that uses **execl(3)** to invoke the command. When a suitable shell is not found, **rwpollexec** uses **/bin/sh**. See BUGS for additional information.

BUGS

rwpollexec uses a subshell to execute the command specified as the argument to **--command**. How the subshell invokes the command is important when the **--timeout** switch is specified. Many shells use a member of **execl(3)** family of functions to invoke the command, which causes the command's process to replace the shell process. For these shells, signals sent by **rwpollexec** are received by the command process directly. However, some shells use a combination of **fork(2)** and **wait(2)** to invoke the command. In these shells, the signal is received by the subshell instead of the command, and this can lead to undesirable or unreliable behavior. When the SILK_RWPOLLEXEC_SHELL environment variable is set, **rwpollexec** uses that shell regardless of how it invokes its command, though if the specified shell uses **fork(2)**, **rwpollexec** will emit a warning to the standard error and to the log. When SILK_RWPOLLEXEC_SHELL is not set, **rwpollexec** attempts to find a shell that uses **execl(3)**. If **rwpollexec** fails to find a suitable shell, it uses */bin/sh* and emits a warning message to standard error and to the log. The list of shells **rwpollexec** checks are specified in the FILES section.

rwpollexec is unable to tell the difference between a command returning a non-zero exit status and a command that fails because the command does not exist or is malformed. Both appear as a failed command with a non-zero exit status. The shell may emit messages that explain why a command has failed. In these instances, these messages will appear in the log.

rwpollexec only attempts to run the command one time. There is no way to tell **rwpollexec** to attempt the command multiple times.

SEE ALSO

silk(7), **kill(1)**, **gzip(1)**, **syslog(3)**, **fork(2)**, **wait(2)**, **execl(3)**

rwreceiver

Accepts files transferred from rwsender(s)

SYNOPSIS

To listen for incoming connections:

```
rwreceiver --mode=server --server-port=[HOST:]PORT
      --client-ident=IDENT [--client-ident=IDENT ...]
      ...
```

To make outgoing connections:

```
rwreceiver --mode=client --server-address=IDENT:HOST:PORT
      [--server-address=IDENT:HOST:PORT ...]
      ...

rwreceiver  SERVER_MODE_OR_CLIENT_MODE_SWITCHES
      --identifier=IDENT --destination-directory=DIR_PATH
      [ --tls-ca=TRUST_FILE
        { { --tls-cert=CERTIFICATE_FILE --tls-key=KEY_FILE }
          | --tls-pkcs12=PKCS12_FILE }
        [--tls-priority=TLS_PRIORITY] [--tls-security=TLS_SECURITY]
        [--tls-crl=CRL_FILE] [--tls-debug-level=DB_LEVEL] ]
      [--post-command=COMMAND]
      [--duplicate-destination=DIR_PATH
        [--duplicate-destination=DIR_PATH...] ]
      [--unique-duplicates]
      [--freespace-minimum=SIZE] [--space-maximum-percent=NUM]
      { --log-destination=DESTINATION
        | --log-pathname=FILE_PATH
        | --log-directory=DIR_PATH [--log-basename=LOG_BASENAME]
          [--log-post-rotate=COMMAND] }
      [--log-level=LEVEL] [--log-sysfacility=NUMBER]
      [--pidfile=FILE_PATH] [--no-chdir] [--no-daemon]

rwreceiver --help

rwreceiver --version
```

DESCRIPTION

rwreceiver is a daemon which accepts files transferred from one or more **rwsender(8)** processes. The received files are stored in a destination directory.

rwreceiver creates multiple copies of the files it receives when one or more **--duplicate-destination** switches are specified. If possible, the duplicate file is created as a reference (a hard link) to the original file. The **--unique-duplicates** switch tells **rwreceiver** not to use hard links and forces **rwreceiver** to create

an individual copy of the file in each duplicate destination directory. Failure to create a file in any of the duplicate destination directories is noted in **rwreceiver**'s log but it is not treated as a failure to transfer the file. Only when a file cannot be created in the destination-directory does **rwreceiver** consider the transfer as failed.

The **--post-command** switch tells **rwreceiver** to execute a command on each file that it successfully receives after the file has been written to the destination directory and copied to each duplicate destination directory. The command may include a placeholder which **rwreceiver** fills with the path to the file in the destination directory. The exit status of the command is ignored by **rwreceiver**. Any output on `stdout` or `stderr` from *COMMAND* normally appears in the log when the log messages are being written to a local log file. See also the **rwpollexec(8)** daemon.

Interaction with rwsender

Either **rwsender** or **rwreceiver** may act as the **server** with the other acting as the **client**. That is, an **rwsender** server may listen for connections from **rwreceiver** clients, or an **rwsender** client may attempt to connect to one or more **rwreceiver** servers.

In addition, each **rwsender** and **rwreceiver** is configured with an identifier of its own and the identifier(s) of the **rwreceiver**(s) or **rwsender**(s) that may connect to it. The connection is closed if the identifier provided by other process is not recognized.

Every **rwsender** that communicates with the same **rwreceiver** must have a unique identifier; likewise, every **rwreceiver** that communicates with the same **rwsender** must have a unique identifier. Ideally, the identifier should provide some information about where the **rwsender** or **rwreceiver** program is running and what sort of data it is transferring.

Disk Usage

By default, if the disk that **rwreceiver** writes to becomes full, **rwreceiver** prints a message to the log file and exits.

To prevent this, specify the **--freespace-minimum** and/or **--space-maximum-percent** switches, which cause **rwreceiver** to monitor its disk usage. These switches were added in SiLK 3.6.

If receiving a file from an **rwsender** process would violate the limits specified in those switches, **rwreceiver** closes the connection to that **rwsender**. This causes the connection to be reestablished, and **rwsender** tries to transfer the file again. If the filesystem is still full, **rwreceiver** closes the connection again. After a delay, the connection is reestablished. This loop is repeated until the file is successfully transferred. The delay between each retry starts at five seconds and grows in five second increments to a maximum of one minute.

When monitoring its disk usage, **rwreceiver** accounts for one copy of the number of bytes in the file. **rwreceiver** does not account for the filesystem overhead associated with creating a file, and it does not consider the space required to create multiple copies of the file (cf., **--duplicate-destination**).

File Creation

The following describes the process **rwreceiver** uses when creating a file it receives from **rwsender**. Administrators may find this information useful when configuring other software to work with **rwreceiver**.

1. **rwsender** sends the name of the file, the size of the file, and the file's permission bits to **rwreceiver**.

2. If a file with that name already exists in **rwreceiver**'s destination directory, **rwreceiver** checks the file's on-disk size. If the size is 0 and no other **rwreceiver** thread is currently handling that file, **rwreceiver** assumes it is an aborted attempt to send the file, and **rwreceiver** removes the existing file. Otherwise, **rwreceiver** tells **rwsender** that the name represents a duplicate file, at which point **rwsender** moves the file to its error directory.
3. When neither **--freespace-minimum** nor **--space-maximum-percent** is specified, processing moves to the next step. Otherwise, **rwreceiver** verifies that there is space on the filesystem to hold one copy of the file. As described in the Disk Usage section above, **rwreceiver** delays processing the file until space is available.
4. **rwreceiver** creates a 0-length placeholder file having the name of the file being transferred, and **rwreceiver** closes this file. The permission bits on this file are all 0.
5. The **rwreceiver** process creates a second file whose name consists of a dot (**.**) followed by the name of the file being transferred. The permission bits on this file are those sent by **rwsender**.
6. **rwreceiver** writes the data it receives from **rwsender** into the dot file.
7. Once the transfer is complete, **rwreceiver** closes the dot file.
8. If any duplicate destination directories have been specified, **rwreceiver** copies the dot file to each of those directories (using a hard link if possible). A failure to copy the file into a duplicate destination is noted in the log file, but otherwise the error is ignored.
9. **rwreceiver** renames the dot file to replace the placeholder file.
10. The **rwreceiver** process tells the **rwsender** process that the transfer was successfully completed.
11. **rwreceiver** prepares the command specified by the **--post-command** switch, perhaps filling in the complete path to the file in the destination directory, and executes the command.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

Application-specific switches

The following set of switches are required:

--identifier=*IDENT*

Use the name *IDENT* when establishing a connection with an **rwsender** process. The identifier should contain only printable, non-whitespace characters; the following characters are illegal: colon (**:**), slash (**/** and \), period (**.**), and comma (**,**).

--mode=*MODE*

Specify how the connection between **rwsender** and **rwreceiver**(s) should be established. When *MODE* is **server**, **rwreceiver** listens for connections from **rwsender** clients; when *MODE* is **client**, **rwreceiver** attempts to connect to **rwsender** servers.

--destination-directory=*DIR_PATH*

Place the transferred files into *DIR_PATH*. Note that **rwreceiver** uses this as its processing directory; see the File Creation section above for details.

Server-mode switches

When running in **server** mode, the following switches are required:

--server-port=[*HOST*:]*PORT*

Listen for incoming **rwsender** client connections on *PORT* as *HOST*. If *HOST* is omitted, **rwreceiver** listens on any address. *HOST* may be a name or an IP address; when *HOST* is an IPv6 address, it must be enclosed in square brackets.

--client-ident=*IDENT*

Allow connections from an **rwsender** client whose identifier is *IDENT*. This switch may be repeated to allow multiple **rwsender** clients to connect. **rwreceiver** closes the connection if an **rwsender** client connects and does not provide a valid identifier.

Client-mode switch

When running in **client** mode, the following switch is required:

--server-address=*IDENT*:*HOST*:*PORT*

Attempt to connect to the **rwsender** server listening to port number *PORT* on the machine *HOST*. **rwreceiver** closes the connection unless the **rwsender** identifies itself as *IDENT*. This switch may be repeated to connect to multiple **rwsender** servers. *HOST* may be a name or an IP address; when *HOST* is an IPv6 address, it must be enclosed in square brackets.

Transport Layer Security switches

It is possible to build SiLK with support for the GnuTLS Transport Layer Security library (https://www.gnutls.org/) which allows **rwsender** and **rwreceiver** to use an encrypted/authenticated channel for their communication. When SiLK includes GnuTLS support, the following switches are available. To enable use of GnuTLS, specify the **--tls-ca** switch and either the **--tls-pkcs12** switch or both the **--tls-cert** and **--tls-key** switches.

--tls-ca=*TRUST_FILE*

Set the trusted certificate authorities to those in *TRUST_FILE*, where *TRUST_FILE* is the complete path to a file containing a PEM-encoded list of certificates. This list of authorities is used to verify the certificate sent by **rwsender**. (Store intermediate certificates in either *CERTIFICATE_FILE* or *PKCS12_FILE*.) This switch must be used in conjunction with either the **--tls-pkcs12** switch or both the **--tls-cert** and the **--tls-key** switches.

--tls-cert=*CERTIFICATE_FILE*

Set the certificate chain (path) for **rwreceiver**'s private key to the list of certificates in *CERTIFICATE_FILE*, where *CERTIFICATE_FILE* is the complete path to a file containing the PEM-encoded certificates. The certificate corresponding to *KEY_FILE* must be listed first; each subsequent intermediate certificate certifies the one before it. This switch may only be used in conjunction with the **--tls-ca** and **--tls-key** switches.

--tls-key=*KEY_FILE*

Read **rwreceiver**'s private encryption key for TLS from *KEY_FILE*, where *KEY_FILE* is the complete path to a PEM-encoded file. The key must not be password protected as there is no support in

rwreceiver for obtaining the password, but note that **--tls-pkcs12** allows a password. This switch may only be used in conjunction with the **--tls-ca** and **--tls-cert** switches.

--tls-pkcs12=*PKCS12_FILE*

Set **rwreceiver**'s encryption certificate and private key for TLS to the contents of *PKCS12_FILE*, where *PKCS12_FILE* is the complete path to a file containing the PKCS#12 contents in DER-format. If intermediate certificates are needed, they should be included in the file. This switch may only be used in conjunction with the **--tls-ca** switch. **rwreceiver** uses the value in the RWRECEIVER_TLS_PASSWORD environment variable to decrypt the PKCS#12 file. If this variable is not set, **rwreceiver** assumes the password is the empty string.

--tls-priority=*TLS_PRIORITY*

Set the preference order (priority) for ciphers, key exchange methods, message authentication codes, and compression methods to those in *TLS_PRIORITY*. This switch is optional; the default value is `NORMAL`. The argument is parsed by the GnuTLS library, and the available arguments depend on the version of GnuTLS linked with SiLK. Detailed information on the format of the argument is available in the GnuTLS documentation under *Priority Strings* (e.g., https://gnutls.org/manual/html_node/Priority-Strings.html) provides the set for the most recent version of GnuTLS; the values used at your site may be different). See also the output of running **gnutls-cli(1)** with the **--priority-list** switch. *Since SiLK 3.18.0.*

--tls-security=*TLS_SECURITY*

Set the security level to use when generating Diffie-Hellman parameters to *TLS_SECURITY*, where *TLS_SECURITY* is one of `low`, `medium`, `high`, or `ultra`. This switch is optional, and when not specified a value of `medium` is used. For the meaning of these values see *Selecting cryptographic key sizes* in the GnuTLS documentation at your site (e.g., https://gnutls.org/manual/html_node/Selecting-cryptographic-key-sizes.html). *Since SiLK 3.18.0.*

--tls-crl=*CRL_FILE*

Update the list of trusted certificates with the certificate revocation lists contained in *CRL_FILE*, where *CRL_FILE* is the complete path to a file containing PEM-encoded list of CRLs. This switch is optional. *Since SiLK 3.18.0.*

--tls-debug-level=*DB_LEVEL*

Set the debugging level used internally by the GnuTLS library to *DB_LEVEL*, an integer between 0 and 99 inclusive. The messages are written to the log destation at the `info` level. The default value of 0 disables debugging. Larger values may reveal sensitive information and should be used carefully. A value above 10 enables all debugging options. *Since SiLK 3.18.0.*

Required logging switches

One of the following logging switches is required:

--log-destination=*DESTINATION*

Specify the destination where logging messages are written. When *DESTINATION* begins with a slash `/`, it is treated as a file system path and all log messages are written to that file; there is no log rotation. When *DESTINATION* does not begin with `/`, it must be one of the following strings:

`none`

Messages are not written anywhere.

`stdout`
: Messages are written to the standard output.

`stderr`
: Messages are written to the standard error.

`syslog`
: Messages are written using the **syslog(3)** facility.

`both`
: Messages are written to the syslog facility and to the standard error (this option is not available on all platforms).

--log-directory=*DIR_PATH*

Use *DIR_PATH* as the directory where the log files are written. *DIR_PATH* must be a complete directory path. The log files have the form

```
DIR_PATH/LOG_BASENAME-YYYYMMDD.log
```

where *YYYYMMDD* is the current date and *LOG_BASENAME* is the application name or the value passed to the **--log-basename** switch when provided. The log files are rotated: At midnight local time, a new log is opened, the previous file is closed, and the command specified by **--log-post-rotate** is invoked on the previous day's log file. (Old log files are not removed by **rwreceiver**; the administrator should use another tool to remove them.) When this switch is provided, a process-ID file (PID) is also written in this directory unless the **--pidfile** switch is provided.

--log-pathname=*FILE_PATH*

Use *FILE_PATH* as the complete path to the log file. The log file is not rotated.

Optional application-specific switches

The following switches are optional:

--post-command=*COMMAND*

Run *COMMAND* on a file once it has been successfully received. The following `%`-conversions are supported in *COMMAND*: `%s` is replaced with the full path of the transferred file in the destination directory, `%I` is replaced with the identifier of the **rwsender** that sent the file, and `%%` is replaced with `%`. If any other character follows `%`, **rwreceiver** exits with an error. Note that *COMMAND* is only invoked on files in the destination directory; however, at the time *COMMAND* is invoked, **rwreceiver** has already copied the file into each of the duplicate destination directories, if any. See also the **rwpollexec(8)** daemon.

--duplicate-destination=*DIR_PATH*

Create a duplicate of each transferred file in the directory *DIR_PATH*. This option may be specified multiple times to create multiple duplicates. This duplicate is made by a hard link to the file in the destination-directory if possible, otherwise a complete copy is made (see also **--unique-duplicates**). If there are errors copying the file to this directory, the error is logged but the process continues as if the transfer was successful. (**rwreceiver** considers a transfer as failed only when the file cannot be created in the destination-directory.)

--unique-duplicates

Force the duplicate file created in each duplicate-destination directory to be a complete copy of the file in the destination-directory instead of a hard link to the file. Using hard links saves disk space and is faster than making a complete copy; however, any modification-in-place to one file affects all files. This switch is ignored when the **--duplicate-destination** switch is not provided.

--freespace-minimum=*SIZE*

Set the minimum amount free space (in bytes) to maintain on the file system where the **--destination-directory** is located. **rwreceiver** delays processing of any file that would cause it to violate this limit (see Disk Usage above). The default value of this switch is 0, which tells **rwreceiver** not to monitor its disk usage. See also **--space-maximum-percent**.

SIZE may be given as an ordinary integer, or as a real number followed by a suffix `K`, `M`, `G`, or `T`, which represents the numerical value multiplied by 1,024 (kilo), 1,048,576 (mega), 1,073,741,824 (giga), and 1,099,511,627,776 (tera), respectively. For example, 1.5K represents 1,536 bytes, or one and one-half kilobytes.

--space-maximum-percent=*NUM*

Use no more than this percentage of the file system containing the **--destination-directory**. The default is to use all of the file system (100%). **rwreceiver** delays processing of files that would cause it to violate this limit. The *NUM* parameter does not need to be an integer. See also **--freespace-minimum** and Disk Usage.

Optional logging and daemon switches

The following are optional switches related to logging and running as a daemon:

--log-level=*LEVEL*

Set the severity of messages that are logged. The levels from most severe to least are: `emerg`, `alert`, `crit`, `err`, `warning`, `notice`, `info`, `debug`. The default is `info`.

--log-sysfacility=*NUMBER*

Set the facility that **syslog(3)** uses for logging messages. This switch takes a number as an argument. The default is a value that corresponds to `LOG_USER` on the system where **rwreceiver** is running. This switch produces an error unless **--log-destination**=syslog is specified.

--log-basename=*LOG_BASENAME*

Use *LOG_BASENAME* in place of the application name in the name of *log* files in the log directory. See the description of the **--log-directory** switch. This switch does **not** affect the name of the process-ID file.

--log-post-rotate=*COMMAND*

Run *COMMAND* on the previous day's log file after log rotation. When this switch is not specified, the previous day's log file is compressed with **gzip(1)**. When the switch is specified and *COMMAND* is the empty string, no action is taken on the log file. Each occurrence of the string `%s` in *COMMAND* is replaced with the full path to the log file, and each occurrence of `%%` is replaced with `%`. If any other character follows `%`, **rwreceiver** exits with an error. Specifying this switch without also using **--log-directory** is an error.

--pidfile=*FILE_PATH*

Set the complete path to the file in which **rwreceiver** writes its process ID (PID) when it is running as a daemon. No PID file is written when **--no-daemon** is given. When this switch is not present, no

PID file is written unless the **--log-directory** switch is specified, in which case the PID is written to *LOGPATH/rwreceiver.pid*.

--no-chdir

Do not change directory to the root directory. When **rwreceiver** becomes a daemon process, it changes its current directory to the root directory so as to avoid potentially running on a mounted file system. Specifying **--no-chdir** prevents this behavior, which may be useful during debugging. The application does not change its directory when **--no-daemon** is given.

--no-daemon

Force **rwreceiver** to run in the foreground---it does not become a daemon process. This may be useful during debugging.

Help switches

The following switches provide help:

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

ENVIRONMENT

RWRECEIVER_TLS_PASSWORD

Specifies the password to use to decrypt the PKCS#12 file specified in the **--tls-pkcs12** switch. When this is not provided, a NULL password is used. Set this environment variable to the empty string for an empty password.

SEE ALSO

rwsender(8), **rwpollexec(8)**, **silk(7)**, **gnutls-cli(1)**, **certtool(1)**, **syslog(3)**, **gzip(1)**, *SiLK Installation Handbook*

rwsender

Watch directory for files and transfer them to rwreceiver(s)

SYNOPSIS

To listen for incoming connections:

```
rwsender --mode=server --server-port=[HOST:]PORT
      --client-ident=IDENT [--client-ident=IDENT ...]
      ...
```

To make outgoing connections:

```
rwsender --mode=client --server-address=IDENT:HOST:PORT
      [--server-address=IDENT:HOST:PORT ...]
      ...

rwsender  SERVER_MODE_OR_CLIENT_MODE_SWITCHES
      --identifier=IDENT --incoming-directory=DIR_PATH
      --processing-directory=DIR_PATH --error-directory=DIR_PATH
      [ --tls-ca=TRUST_FILE
        { { --tls-cert=CERTIFICATE_FILE --tls-key=KEY_FILE }
          | --tls-pkcs12=PKCS12_FILE }
        [--tls-priority=TLS_PRIORITY] [--tls-security=TLS_SECURITY]
        [--tls-crl=CRL_FILE] [--tls-debug-level=DB_LEVEL] ]
      [--local-directory=[[IDENT]:]DIR_PATH
        [--local-directory=[[IDENT]:]DIR_PATH ...]]
      [--unique-local-copies]
      [--filter=IDENT:REGEXP] [--priority=NUM:REGEXP]
      [--polling-interval=NUM]
      [--send-attempts=NUM] [--block-size=NUM]
      { --log-destination=DESTINATION
        | --log-pathname=FILE_PATH
        | --log-directory=DIR_PATH [--log-basename=LOG_BASENAME]
          [--log-post-rotate=COMMAND] }
      [--log-level=LEVEL] [--log-sysfacility=NUMBER]
      [--pidfile=FILE_PATH] [--no-chdir] [--no-daemon]

rwsender --help

rwsender --version
```

DESCRIPTION

rwsender is a daemon which watches an incoming directory for files. As files are added to the incoming directory, they are moved into a processing directory and then transferred over the network to one or more **rwreceiver(8)** processes. Files in the incoming directory may also be "transferred" to one or more local directories.

As **rwsender** scans the incoming directory, it ignores a file if its size is 0 bytes or if its name begins with a dot (`.`). On each scan, if **rwsender** detects a file name that was not present in the previous scan, it records the name and size of the file. If the file has a different size on the next scan, the new size is recorded. Once the file has the same size on two consecutive scans, **rwsender** moves the file to the processing directory and queues it for transfer.

Interaction with rwreceiver

Either **rwsender** or **rwreceiver** may act as the **server** with the other acting as the **client**. That is, an **rwsender** server may listen for connections from **rwreceiver** clients, or an **rwsender** client may attempt to connect to one or more **rwreceiver** servers.

In addition, each **rwsender** and **rwreceiver** is configured with an identifier of its own and the identifier(s) of the **rwreceiver**(s) or **rwsender**(s) that may connect to it. The connection is closed if the identifier provided by other process is not recognized.

Every **rwsender** that communicates with the same **rwreceiver** must have a unique identifier; likewise, every **rwreceiver** that communicates with the same **rwsender** must have a unique identifier. Ideally, the identifier should provide some information about where the **rwsender** or **rwreceiver** program is running and what sort of data it is transferring.

OPTIONS

Option names may be abbreviated if the abbreviation is unique or is an exact match for an option. A parameter to an option may be specified as **--arg**=*param* or **--arg** *param*, though the first form is required for options that take optional parameters.

Application-specific switches

The following set of switches are required:

--identifier=*IDENT*

> Use the name *IDENT* when establishing a connection with an **rwreceiver** process. The identifier should contain only printable, non-whitespace characters; the following characters are illegal: colon (`:`), slash (`/` and `\`), period (`.`), and comma (`,`).

--mode=*MODE*

> Specify how the connection between **rwsender** and **rwreceiver**(s) should be established. When *MODE* is **server**, **rwsender** listens for connections from **rwreceiver** clients; when *MODE* is **client**, **rwsender** attempts to connect to **rwreceiver** servers.

--incoming-directory=*DIR_PATH*

> Periodically scan the directory *DIR_PATH* for files to transfer. As **rwsender** scans *DIR_PATH*, it ignores a file if its name begins with a dot (`.`) or if its size is 0 bytes. When a file is first detected, its size is recorded, and the file must have the same size for two consecutive scans before **rwsender** will add it to sending queue. The interval between scans is set by **--polling-interval**. *DIR_PATH* must be a complete directory path.

--processing-directory=*DIR_PATH*

Use *DIR_PATH* as a location to cache files until they are successfully transferred. For each **rwreceiver** *IDENT* specified on the command line, a subdirectory is created under *DIR_PATH* to hold a copy of each file that is to be sent to that **rwreceiver**. (**rwsender** uses a reference (a hard link) to the file instead of a copy of the file when possible.) *DIR_PATH* must be a complete directory path.

--error-directory=*DIR_PATH*

Move a file that is rejected by an **rwreceiver** (for example, because it has a duplicate filename) to the subdirectory *IDENT* of *DIR_PATH*, where *IDENT* is the identifier of the **rwreceiver** that rejected the file. *DIR_PATH* must be a complete directory path.

Server-mode switches

When running in **server** mode, the following switches are required:

--server-port=[*HOST*:]*PORT*

Listen for incoming **rwreceiver** client connections on *PORT* as *HOST*. If *HOST* is omitted, **rwsender** listens on any address. *HOST* may be a name or an IP address; when *HOST* is an IPv6 address, it must be enclosed in square brackets.

--client-ident=*IDENT*

Allow connections from an **rwreceiver** client whose identifier is *IDENT*. This switch may be repeated to allow multiple **rwreceiver** clients to connect. **rwsender** closes the connection if an **rwreceiver** client connects and does not provide a valid identifier.

Client-mode switch

When running in **client** mode, the following switch is required:

--server-address=*IDENT*:*HOST*:*PORT*

Attempt to connect to the **rwreceiver** server listening to port number *PORT* on the machine *HOST*. **rwsender** closes the connection unless the **rwreceiver** identifies itself as *IDENT*. This switch may be repeated to connect to multiple **rwreceiver** servers. *HOST* may be a name or an IP address; when *HOST* is an IPv6 address, it must be enclosed in square brackets.

Transport Layer Security switches

It is possible to build SiLK with support for the GnuTLS Transport Layer Security library (https://www.gnutls.org/) which allows **rwsender** and **rwreceiver** to use an encrypted/authenticated channel for their communication. When SiLK includes GnuTLS support, the following switches are available. To enable use of GnuTLS, specify the **--tls-ca** switch and either the **--tls-pkcs12** switch or both the **--tls-cert** and **--tls-key** switches.

--tls-ca=*TRUST_FILE*

Set the trusted certificate authorities to those in *TRUST_FILE*, where *TRUST_FILE* is the complete path to a file containing a PEM-encoded list of certificates. This list of authorities is used to verify the certificate sent by **rwreceiver**. (Store intermediate certificates in either *CERTIFICATE_FILE* or *PKCS12_FILE*.) This switch must be used in conjunction with either the **--tls-pkcs12** switch or both the **--tls-cert** and the **--tls-key** switches.

--tls-cert=*CERTIFICATE_FILE*

Set the certificate chain (path) for **rwsender**'s private key to the list of certificates in *CERTIFICATE_FILE*, where *CERTIFICATE_FILE* is the complete path to a file containing the PEM-encoded certificates. The certificate corresponding to *KEY_FILE* must be listed first; each subsequent intermediate certificate certifies the one before it. This switch may only be used in conjunction with the **--tls-ca** and **--tls-key** switches.

--tls-key=*KEY_FILE*

Read **rwsender**'s private encryption key for TLS from *KEY_FILE*, where *KEY_FILE* is the complete path to a PEM-encoded file. The key must not be password protected as there is no support in **rwsender** for obtaining the password, but note that **--tls-pkcs12** allows a password. This switch may only be used in conjunction with the **--tls-ca** and **--tls-cert** switches.

--tls-pkcs12=*PKCS12_FILE*

Set **rwsender**'s encryption certificate and private key for TLS to the contents of *PKCS12_FILE*, where *PKCS12_FILE* is the complete path to a file containing the PKCS#12 contents in DER-format. If intermediate certificates are needed, they should be included in the file. This switch may only be used in conjunction with the **--tls-ca** switch. **rwsender** uses the value in the RWSENDER_TLS_PASSWORD environment variable to decrypt the PKCS#12 file. If this variable is not set, **rwsender** assumes the password is the empty string.

--tls-priority=*TLS_PRIORITY*

Set the preference order (priority) for ciphers, key exchange methods, message authentication codes, and compression methods to those in *TLS_PRIORITY*. This switch is optional; the default value is `NORMAL`. The argument is parsed by the GnuTLS library, and the available arguments depend on the version of GnuTLS linked with SiLK. Detailed information on the format of the argument is available in the GnuTLS documentation under *Priority Strings* (e.g., https://gnutls.org/manual/html_node/Priority-Strings.html) provides the set for the most recent version of GnuTLS; the values used at your site may be different). See also the output of running **gnutls-cli(1)** with the **--priority-list** switch. *Since SiLK 3.18.0.*

--tls-security=*TLS_SECURITY*

Set the security level to use when generating Diffie-Hellman parameters to *TLS_SECURITY*, where *TLS_SECURITY* is one of `low`, `medium`, `high`, or `ultra`. This switch is optional, and when not specified a value of `medium` is used. For the meaning of these values see *Selecting cryptographic key sizes* in the GnuTLS documentation at your site (e.g., https://gnutls.org/manual/html_node/Selecting-cryptographic-key-sizes.html). *Since SiLK 3.18.0.*

--tls-crl=*CRL_FILE*

Update the list of trusted certificates with the certificate revocation lists contained in *CRL_FILE*, where *CRL_FILE* is the complete path to a file containing PEM-encoded list of CRLs. This switch is optional. *Since SiLK 3.18.0.*

--tls-debug-level=*DB_LEVEL*

Set the debugging level used internally by the GnuTLS library to *DB_LEVEL*, an integer between 0 and 99 inclusive. The messages are written to the log destation at the `info` level. The default value of 0 disables debugging. Larger values may reveal sensitive information and should be used carefully. A value above 10 enables all debugging options. *Since SiLK 3.18.0.*

Required logging switches

One of the following logging switches is required:

--log-destination=*DESTINATION*

Specify the destination where logging messages are written. When *DESTINATION* begins with a slash `/`, it is treated as a file system path and all log messages are written to that file; there is no log rotation. When *DESTINATION* does not begin with `/`, it must be one of the following strings:

`none`
: Messages are not written anywhere.

`stdout`
: Messages are written to the standard output.

`stderr`
: Messages are written to the standard error.

`syslog`
: Messages are written using the **syslog(3)** facility.

`both`
: Messages are written to the syslog facility and to the standard error (this option is not available on all platforms).

--log-directory=*DIR_PATH*

Use *DIR_PATH* as the directory where the log files are written. *DIR_PATH* must be a complete directory path. The log files have the form

```
DIR_PATH/LOG_BASENAME-YYYYMMDD.log
```

where *YYYYMMDD* is the current date and *LOG_BASENAME* is the application name or the value passed to the **--log-basename** switch when provided. The log files are rotated: At midnight local time, a new log is opened, the previous file is closed, and the command specified by **--log-post-rotate** is invoked on the previous day's log file. (Old log files are not removed by **rwsender**; the administrator should use another tool to remove them.) When this switch is provided, a process-ID file (PID) is also written in this directory unless the **--pidfile** switch is provided.

--log-pathname=*FILE_PATH*

Use *FILE_PATH* as the complete path to the log file. The log file is not rotated.

Optional application-specific switches

These are application-specific switches that are not required:

--local-directory=[[*IDENT*]:]*DIR_PATH*

Create a duplicate of each incoming file in the directory *DIR_PATH*. This switch may be specified multiple times to create multiple duplicates. The duplicate is made by a reference (a hard link) to the file in the processing-directory if possible, otherwise a complete copy is made. (Note that any modification-in-place to a file reference affects all references to that file; use **--unique-local-copies** to avoid this). If *IDENT* is specified, filters may be used to determine which files get copied to *DIR_PATH*. See --filter=*IDENT*:*REGEXP*"> for filter details. When *DIR_PATH* contains the colon character and no *IDENT* is wanted, a single colon may precede *DIR_PATH* to designate an empty *IDENT*.

--unique-local-copies

Force the duplicate file created in each local-directory to be a complete copy of the file in the processing-directory instead of a reference (a hard link) to the file. Using references saves disk space and is faster than making a complete copy; however, any modification-in-place to one file affects all files. **rwsender** always makes a complete copy when it is unable to make a reference. This switch is ignored when the **--local-directory** switch is not provided.

--filter=*IDENT*:*REGEXP*

Configure **rwsender** to transfer files matching the regular expression *REGEXP* to the **rwreceiver** whose identifier is *IDENT*, or to copy files to the local directory labeled as *IDENT*. This switch may be repeated. When this switch is not provided, all **rwreceiver**s and local directories get all files. When this switch is provided, any files not matching a *REGEXP* are left in the incoming directory and are not transferred.

The regular expression must be a POSIX 1003.2 *modern* or *extended* regular expressions, roughly akin to those used by **egrep(1)**. Documentation might be found in the **regex(7)** or **re_format(7)** manual pages on your system.

The filter is only applied to files in the incoming-directory. Once a file has been moved into an **rwreceiver**-specific subdirectory of the processing-directory, restarting **rwsender** with a different set of **--filter** switches does not affect the files previously queued for each **rwreceiver**. To apply the filters to unsent files, you must stop the **rwsender** process, move all files from the subdirectories of the processing-directory to the incoming-directory, and restart the **rwsender** process.

--priority=*NUM*:*REGEXP*

Set the priority of files that match *REGEXP* to *NUM*. *NUM* must be an integer between 0 and 100 inclusive. In the current version of **rwsender**, priorities 0 through 50 get grouped into a single *low* priority bin, and priorities 51 through 100 get grouped into a single *high* priority bin. Files in the high priority bin are generally be sent before files in the low priority bin. The default priority of a file is 50. This switch may be repeated for multiple priorities.

--polling-interval=*NUM*

Configure **rwsender** to check the incoming directory for new files every *NUM* seconds. The default polling interval is 15 seconds.

--send-attempts=*NUM*

For each file going to an **rwreceiver**, make *NUM* attempts to open the file, map its contents, and send the contents to that **rwreceiver**. After *NUM* attempts, the file is ignored by **rwsender** but the file remains in the **rwreceiver**-specific subdirectory of the processing directory. Unless the file is manually removed from the processing directory, **rwsender** again attempts to send the file when **rwsender** is restarted. The limit may be set to a value from 1 to 65535. When *NUM* is 0, there is no limit. The default number of attempts is 5.

--block-size=*NUM*

Specify the chunk size in bytes that **rwsender** uses when sending files to **rwreceiver**s. The default number of bytes is 8192; the valid range is 256 to 65535.

Optional logging and daemon switches

The following are optional switches related to logging and running as a daemon:

--log-level=*LEVEL*

Set the severity of messages that are logged. The levels from most severe to least are: `emerg`, `alert`, `crit`, `err`, `warning`, `notice`, `info`, `debug`. The default is `info`.

--log-sysfacility=*NUMBER*

Set the facility that **syslog(3)** uses for logging messages. This switch takes a number as an argument. The default is a value that corresponds to `LOG_USER` on the system where **rwsender** is running. This switch produces an error unless **--log-destination**=syslog is specified.

--log-basename=*LOG_BASENAME*

Use *LOG_BASENAME* in place of the application name in the name of *log* files in the log directory. See the description of the **--log-directory** switch. This switch does **not** affect the name of the process-ID file.

--log-post-rotate=*COMMAND*

Run *COMMAND* on the previous day's log file after log rotation. When this switch is not specified, the previous day's log file is compressed with **gzip(1)**. When the switch is specified and *COMMAND* is the empty string, no action is taken on the log file. Each occurrence of the string `%s` in *COMMAND* is replaced with the full path to the log file, and each occurrence of `%%` is replaced with `%`. If any other character follows `%`, **rwsender** exits with an error. Specifying this switch without also using **--log-directory** is an error.

--pidfile=*FILE_PATH*

Set the complete path to the file in which **rwsender** writes its process ID (PID) when it is running as a daemon. No PID file is written when **--no-daemon** is given. When this switch is not present, no PID file is written unless the **--log-directory** switch is specified, in which case the PID is written to *LOGPATH/rwsender.pid*.

--no-chdir

Do not change directory to the root directory. When **rwsender** becomes a daemon process, it changes its current directory to the root directory so as to avoid potentially running on a mounted file system. Specifying **--no-chdir** prevents this behavior, which may be useful during debugging. The application does not change its directory when **--no-daemon** is given.

--no-daemon

Force **rwsender** to run in the foreground---it does not become a daemon process. This may be useful during debugging.

Help switches

The following switches provide help:

--help

Print the available options and exit.

--version

Print the version number and information about how SiLK was configured, then exit the application.

ENVIRONMENT

RWSENDER_TLS_PASSWORD

Specifies the password to use to decrypt the PKCS#12 file specified in the **--tls-pkcs12** switch. When this is not provided, a NULL password is used. Set this environment variable to the empty string for an empty password.

SEE ALSO

rwreceiver(8), **silk(7)**, **gnutls-cli(1)**, **certtool(1)**, **syslog(3)**, **egrep(1)**, **gzip(1)**, **regex(7)**, **re_format(7)**, *SiLK Installation Handbook*

BUGS

An attempt should be made to use a unique name for each file put into the incoming directory. When a file is added to the incoming directory that has the same name as a file in the processing directory, the file added to the incoming directory replaces the existing file in the processing directory.

Appendix A

License

SiLK 3.22.0

Copyright 2023 Carnegie Mellon University.

GNU GPL 2.0

June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.

51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its

recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by

you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to

refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

One line to give the program's name and a brief idea of what it does.

Copyright (C) ¡year¿ ¡name of author¿

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.

signature of Ty Coon, 1 April 1989

Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

SiLK 3.22.0 includes and/or can make use of certain third party software ("Third Party Software"). The Third Party Software that is used by SiLK 3.22.0 is dependent upon your system configuration, but typically includes the software identified in this license.txt file, and/or described in the documentation and/or read me file. By using SiLK 3.22.0, You agree to comply with any and all relevant Third Party Software terms and conditions contained in any such Third Party Software or separate license file distributed with such Third Party Software. The parties who own the Third Party Software ("Third Party Licensors") are intended third party beneficiaries to this License with respect to the terms applicable to their Third Party Software. Third

Party Software licenses only apply to the Third Party Software and not any other portion of SiLK 3.22.0 or SiLK 3.22.0 as a whole.

This material is based upon work funded and supported by the Department of Homeland Security under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

GOVERNMENT PURPOSE RIGHTS – Software and Software Documentation

Contract No.: FA8702-15-D-0002 Contractor Name: Carnegie Mellon University Contractor Address: 4500 Fifth Avenue, Pittsburgh, PA 15213

The Government's rights to use, modify, reproduce, release, perform, display, or disclose this software are restricted by paragraph (b)(2) of the Rights in Noncommercial Computer Software and Noncommercial Computer Software Documentation clause contained in the above identified contract. No restrictions apply after the expiration date shown above. Any reproduction of the software or portions thereof marked with this legend must also reproduce the markings.

Carnegie Mellon® and CERT® are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM23-0973