Tivoli Netcool Supports Guide to the Tivoli EIF probe

by

Jim Hutchinson

Document release: 3.1

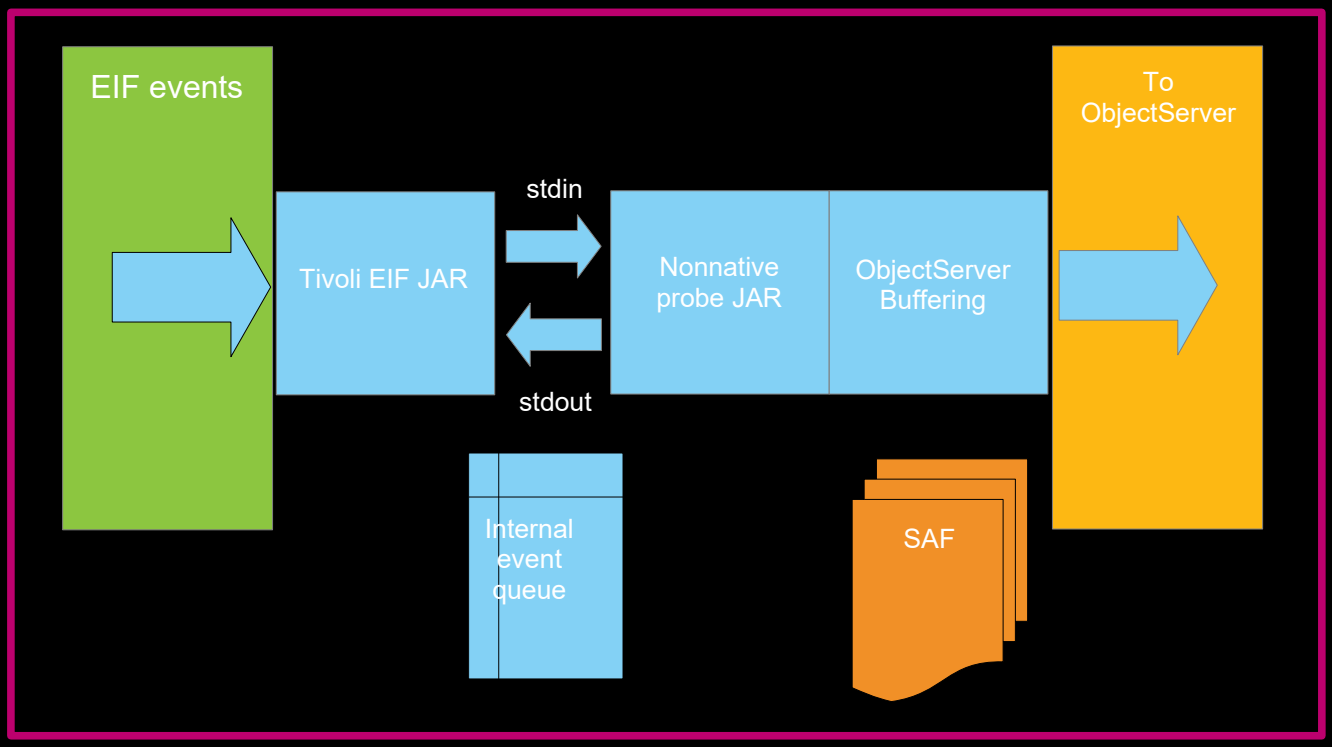
Table of Contents

1 Introduction

1.1 Overview

The Tivoli EIF probe is a Java based socket listener probe that uses the Non-Native Java probe to send events to the object server. EIF events are forwarded to the probes defined socket using IBM Tivoli software such as posteifmsg, ITM etc. The latest version of the probe removed the use of a EIF cache file, to improve performance. The Tivoli EIF probe is now capable of managing up to five times the load of the earlier probes, provided it is given enough CPU and memory resources.

Event Processing Diagram



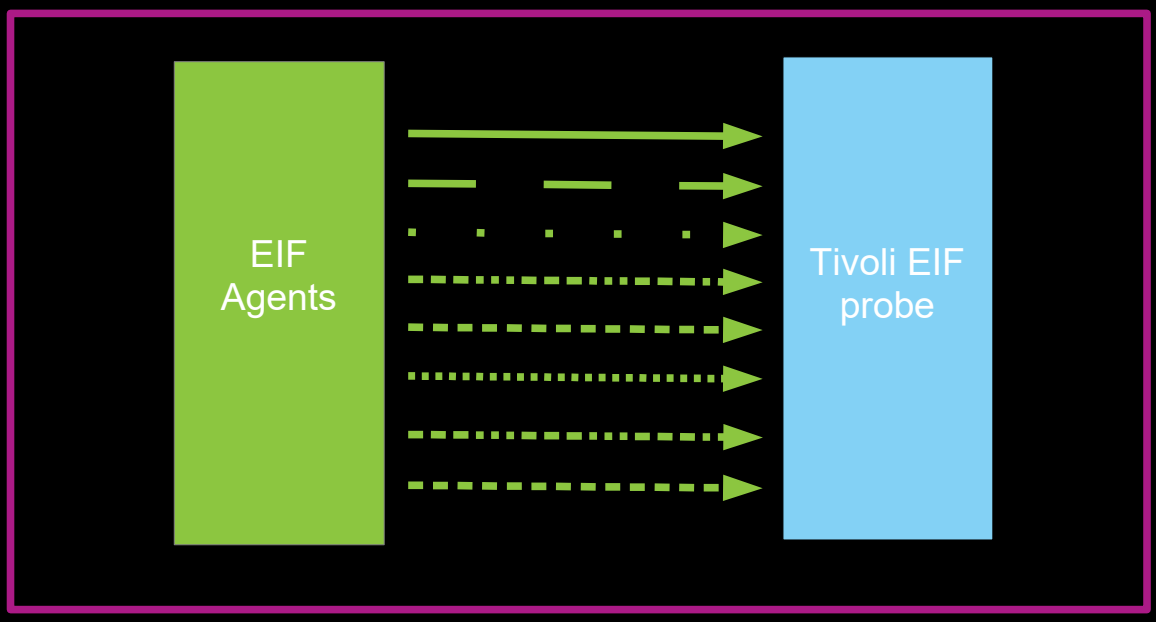
1.2 EIF Connection processing

There are a number of factors that affect the probes ability to manage clients that connect to it. Typically the probe will be able to manage large numbers of connections. One of the most obvious throttle parameters is the number of open files the probe user is allocated. It is generally best to use the largest setting the operating system allows, and tune the operating system as a Web server.
The Tivoli EIF probe is susceptible to misbehaving clients, as these can drain or take up connections, preventing active clients from sending their data. It is important to understand which clients are connecting to the probe and how they can affect the probes behaviour. If required use TCPDUMP to monitor the client activity on the probes port.
The probe performs a periodic self-test called an EIFHeartBeat. This test involves attempting to connect to the probes port, as a EIF agent. The number of failed connection attempts if stored, with the number of heartbeat failures, when set, causing the probe to exit.
e.g.
EIFHeartBeatInterval: 30
EIFHeartBeatFailures: 2

The example settings do not cause the probe to exit after 60 seconds (2*30), but the probe will exit after 2 heartbeat failures, which will be after at least 60 seconds.

The command 'netstat –na' can be used to monitor the number of connections made to the probes port.

Clients can connect using a permanent connection, or else connect as and when required. For clients that send large volumes of data a permanent connection is most efficient.



1.3 Probe version

The latest probe version is Release ID: 13.3.0.

```
Netcool/OMNIbus tivoli_eif probe - Version 8.1.0 64-bit
(C) Copyright IBM Corp. 1994, 2012

Netcool/OMNIbus Probe API Library Version 8.1.0 64-bit
Release ID: 13.3.0
JAR Build Date: Wed Dec 09 2015 19:44:19 on rhat3es-build1 (Linux 2.4.21-
47.0.1.ELsmp)
API Release ID: 5.50.86
Library Revisions:
        libnetcool: 5.50.86
        network::ipv6: 5.50.20
Software Compile Date: Wed Aug 15 07:15:36 UTC 2018 on rhat5es-build1.hursley.ibm.com
(Linux 2.6.18-274.17.1.el5 #1 SMP Wed Jan 4 22:45:44 EST 2012)
```

Check the probes environment file to make sure the JAR files are correct.

File : $NCHOME/omnibus/probes/java/nco_p_tivoli_eif.env

EIF_JARS=${OMNIHOME}/java/jars/evd.jar:${OMNIHOME}/java/jars/log.jar

```
ls -l ${OMNIHOME}/java/jars/log.jar
139,027
ls -l ${OMNIHOME}/java/jars/evd.jar
224,742
```

You can use the probes environment file to set useful variables.

```
# Custom settings
#
# Number of open files
ulimit -n 2048
# Netcool/OMNIbus locale settings
LANG=en_GB.utf8
LC_ALL=en_GB.utf8
export LANG LC_ALL
# Debug messages seen with -version
ulimit -n
echo "LANG=$LANG"
echo "LC_ALL=$LC_ALL"
# EOF
```

2 Probe Properties

The following Tivoli EIF probe specific properties are available;

```
# Specific Properties
#
# HandleMalformedAlarms          : "true"
# EIFHeartBeatInterval           : 0
# EIFHeartBeatFailures           : 0
# EIFTraceFileName               : '$OMNIHOME/var/tivoli_eif.trc'
# EIFTraceLevel                  : 'NO'
# EIFDebugFileName               : '$OMNIHOME/var/debug_eif.trc'
# EIFDebugLevel                  : 'NO'
# EIFLogFileName                 : '$OMNIHOME/var/tivoli_eif.log'
# EIFLogLevel                    : 'NO'
# EventCopies                    : 1
# MaxEventQueueSize              : 10000
# PortMapper                     : "false"
# PortMapperNumber               : 100033057
# PortNumber                     : 9998
# EIFReadRetryInterval           : 120
# StreamCapture                  : "false"
# StreamCaptureFile              : '$OMNIHOME/var/tivoli_eif.stream'
```

2.1 MaxEventQueueSize

When the event queue is full [MaxEventQueueSize is reached] new EIF events are discarded by the Tivoli EIF probe.

2.2 EIFHeartBeatInterval

The EIFHeartBeatInterval property allows the probe to check that its port is still available and that new EIF events can be processed.

2.3 EIFHeartBeatFailures

The EIFHeartBeatFailures property allows the probe to exit after a given number of probe heartbeat failures. It is not directly related the EIFHeartBeatInterval property, since the probe must perform other tasks, as well as perform heartbeats. Normally EIFHeartBeatFailures can be left unset.

2.4 EIFReadRetryInterval

The EIFReadRetryInterval property allows the user to define how long the probe waits for an event to complete before retrying to read the event. This is useful when the EIF sender is unresponsive or sends partial event data.

2.5 Example property settings

The following settings are for a typical Tivoli EIF probe connecting to the collection layer in a multitier system:

```
# Object Server settings
Server                  : 'PCOL_P'
ServerBackup            : 'PCOL_B'
# PollServer > NetworkTimeout
PollServer              : 30
NetworkTimeout          : 15
# Buffering enabled with a maximum 10 second delay
Buffering               : 1
BufferSize              : 500
FlushBufferInterval     : 9
# EIF Settings
MaxEventQueueSize       : 100000
PortNumber              : 12345
# Allowing recovery from stalled port - uncomment when needed
# EIFHeartBeatInterval    : 30
# EIFHeartBeatFailures    : 2
#EOF
```

Note: Set NetworkTimeout to a value other 0 to ensure ObjectServer timout occurs even when only a single ObjectServer is connected to, otherwise there is a risk that the ObjectServer disconnection is never detected.

3 Process Automation

The Tivoli EIF probe runs under process but it may experience issues when under heavy load. Process control may attempt to kill the process unnecessarily. It is therefore necessary to allow the probe more time to respond to process control requests.

On UNIX use:

```
nco_pad -retrytime  120  -roguetimeout 120
```

Where:
-retrytime <value> : Failed process start retry time-out.
-roguetimeout <value> : Allowed time in seconds for proce

With the probe process being defined as usual:

```
nco_process 'TivoliEIFProbe'
{
    Command '$OMNIHOME/probes/nco_p_tivoli_eif -propsfile $OMNIHOME/probes/solaris2/tivoli_eif.props' run as 0
    Host            =       'localhost'
    Managed         =       True
    RestartMsg      =       '${NAME} running as ${EUID} has been restored on ${HOST}.'
    AlertMsg        =       '${NAME} running as ${EUID} has died on ${HOST}.'
    RetryCount      =       0
    ProcessType     =       PaPA_AWARE
}
```

The important property settings are EIFHeartBeatFailures and EIFHeartBeatInterval.
The probe should be configured to exit if there are one or more heart beat failures:
e.g.
EIFHeartBeatInterval : 30
EIFHeartBeatFailures : 2

The minimum time the probe will exit after is therefore 60 seconds, with the maximum time being defined by network settings.

3.1 NCO_PA.props

The NCO_PA.props file can be used in Netcool/OMNIbus 8.1 to define the rogue and retry timeouts.

```
Authenticate: 'NONE'
Connections: 100
Name: 'NCO_PA'
ConfigFile: '$NCHOME/omnibus/etc/NCO_PA.conf'
MessageLog: '$OMNIHOME/log/NCO_PA.log'
PidFile: './var/NCO_PA.pid'
# Retry and Rogue timeouts
RetryTime: 30
RogueTimeout: 60
KillProcessGroup: TRUE
# End of File
```

4 Managing EIF loads

The Tivoli EIF probe is designed to accept as many connections as the operating system can handle. In this respect, the probe server can be tuned to the same parameters that a Web server is tuned. The difference is that the Tivoli EIF probe runs in two parts; the listener socket, and the object server based event processing. In order to allow the latter to keep up with the event load, it is important to monitor the processes CPU usage and know the maximum sustained event loads that the probe processes can handle. If only permanent connections are being made to the probe, then the probe server can be configured to accept the required connections plus a percentage to allow for re-connections, and connection issues.

4.1 Peer-to-Peer

The Tivoli EIF probe supports Peer-to-Peer, and in this scenario two identical EIF events sources must be configured to forward events to the master and slave probes. EIF agents should not be configured to failover to a slave probe, as the events will be discarded. For EIF agent failover, a backup or secondary probe should be used. Customers prefer to a pool of Tivoli EIF probes and a load balancer, rather than Peer-To-Peer where many EIF agents are connecting randomly.

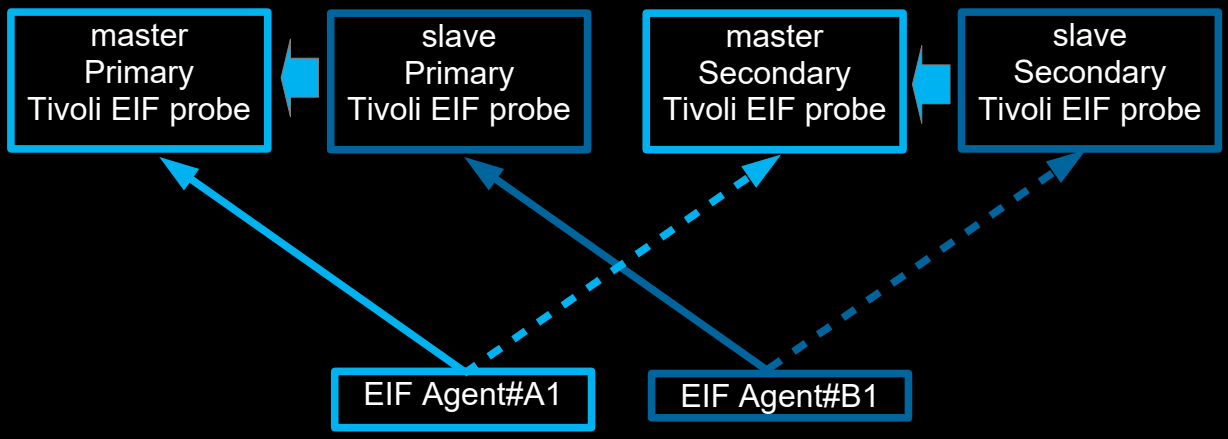
4.2 Load Balancing

There is not an automatic way to perform load balancing for the EIF probe. The best was to balance the event load is to test the probes behaviour using the expected maximum and normal loads. For example if there are a known number and type of EIF agents, their normal load and expected maximum loading can be estimated. This can then be used to exercise the Tivoli EIF probe on the probe server platform to determine its behaviour then perform the necessary tuning and load balancing.

The number of Tivoli EIF probes required to collect a networks EIF data will depend upon the maximum event rate the operating system can sustain, and the maximum volume of events the EIF agents could possibly send. Spreading the load over a number of probes and collection object servers is recommended, as using a single Tivoli EIF probe does not provide any resilience and limits the throughput of EIF events to the maximum possible of the given probe server.

Example Dual Resilient load balanced system using Peer-to-Peer

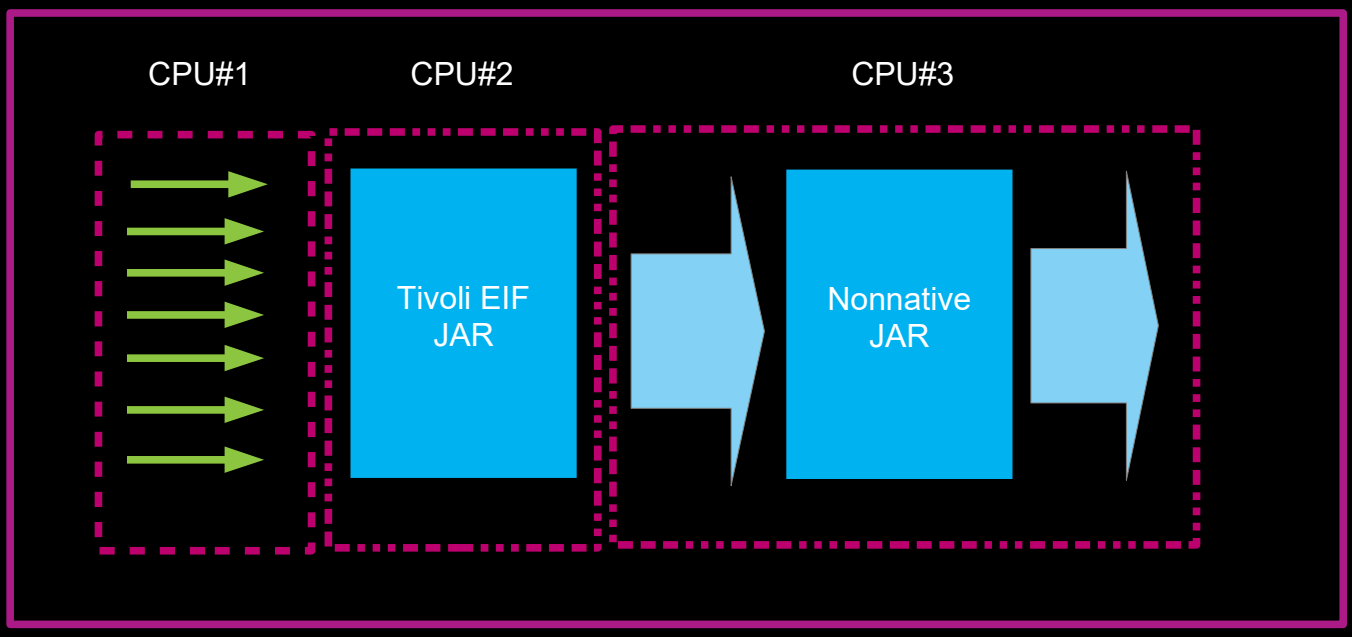
EIF Agents [A] send data to the master probes
EIF Agents [B] send the same data to the slave probes
Each probe server should ideally be on a unique physical server



4.3 Understanding loading

The Tivoli EIF probe is made up of two distinct processes. It is therefore important to understand the affect loading has on each of them. Whilst the Tivoli EIF JAR listens for incoming data, the non-native probe processes the data and forwards it to the object server. Under excessive loads, the Tivoli EIF JAR can consume the majority of the CPU, preventing the non-native probe from processing the data successfully. It this therefore important to monitor both processes and adjust the probe servers resources accordingly. If sustained high loads are normal, it is best to split the load over two or more probe servers.

Because the operating system needs to spend time managing the incoming connections, the probe server should have at least four CPU's/Cores where loading is expected.



To monitor the probe successfully, each probe process should be monitored alongside the CPU usage of the system.

5 Replaying Stream Capture files

The Tivoli EIF probe allows the event stream to be captured in a StreamCaptureFile. This file can be replayed to the probe using the posteifmsg program provided with the EIF SDK. In order to replay the stream capture file, the data must be reformatted to meet with the syntax used by posteifmsg. The script create_postemsg is provided here for this purpose. The script has the additional feature of numbering each event so that they can be identified within the Tivoli EIF probes debug log and rules file. The script uses GNU awk, as long strings are usually found in stream capture files. If this is not the case, awk can be used. The commands gawk or awk, and posteifmsg need to be in the users path for the script to run.

5.1 Example EIF.conf

```
ServerLocation=localhost
ServerPort=4444
# Other settings
BufferEvents=NO
RetryInterval=5
BufEvtPath=/tmp/test_eif.cache
LogFileName=/tmp/test_eif.log
#EOF
```

5.2 create_postemsgs Script

```
#! /bin/sh
if [ $# -ne 1 ]
then
 echo "Usage : `basename $0` [streamcapture file]"
 exit
fi
cat $1 | \
gawk -F\; '{ for (i=1;i<=NF;++i)\
if($i!="){\
if($i!="){\
if(i==1){printf "posteifmsg -f EIF.conf "} else
if($i=="END"){printf "TESTING TEST_%s\n",NR}\
else { printf "%s ",$i }
}\
}\
}'
echo
#EOF
```

5.3 Replication Steps

- Start probe on localhost

```
$OMNIHOME/probes/nco_p_tivoli_eif -port 4444
```

- Create postemsg commands from stream capture

```
create_postemsgs > run_probe_test
```

- Run test to probe

```
./run_probe_test
```

Where

- tivoli_eif.stream file is the event stream file from a Tivoli EIF probe
- posteifmsg is the posteifmsg binary available with the Tivoli EIF SDK package
- EIF.conf is the configuration file for posteifmsg

6 Appendix

6.1 Solaris Performance Tuning

Solaris TCP_KEEPALIVE_INTERVAL

Description:

The keepAlive packet ensures that a connection stays in an active and established state.

How to view or set:

Use the ndd command to determine the current value or to set the value.

For example:

```
ndd -set /dev/tcp tcp_keepalive_interval 300000
```

```
Default value: 7200000 milliseconds
Recommended value: 15000 milliseconds
```

Solaris TCP_TIME_WAIT_INTERVAL

Description:

Notifies TCP/IP on how long to keep the connection control blocks closed. After the applications complete the TCP/IP connection, the control blocks are kept for the specified time. When high connection rates occur, a large backlog of the TCP/IP connections accumulates and can slow server performance. The server can stall during certain peak periods. If the server stalls, the netstat command shows that many of the sockets that are opened to the HTTP server are in the CLOSE_WAIT or FIN_WAIT_2 state. Visible delays can occur for up to four minutes, during which time the server does not send any responses, but CPU utilization stays high, with all of the activities in system processes.

How to view or set:

Use the get command to determine the current interval and the set command to specify an interval of 30 seconds.

For example:

```
ndd -get /dev/tcp tcp_time_wait_interval
ndd -set /dev/tcp tcp_time_wait_interval 30000
```

Default value: The default time wait interval for a Solaris operating system is 240000 milliseconds, which is equal to 4 minutes.

Recommended value: 60000 milliseconds

Tivoli EIF Probe specific:

```
Recommended TCP_KEEPALIVE_INTERVAL : 30-120 seconds
Recommended TCP_TIME_WAIT_INTERVAL : 30-120 seconds
```

To see all parameters:

```
ndd /dev/tcp \?
```

6.1.1 Minimising the probes recovery time

The tcp_ip_abort_cinterval setting affects how quickly the probe can recover from the CLOSE_WAIT issue. The minimum recommended setting is 60 seconds.

```
ndd -set /dev/tcp tcp_ip_abort_cinterval 60000
```

6.1.2 Example Solaris settings

In the probe users environment ensure limits are set:

```
ulimit -n 4096
ulimit -s 1024
```

When the system is started, set the network device settings:

```
/usr/sbin/ndd -set /dev/tcp tcp_keepalive_interval 15000
/usr/sbin/ndd -set /dev/tcp tcp_time_wait_interval 60000

/usr/sbin/ndd -set /dev/tcp tcp_ip_abort_cinterval 60000
/usr/sbin/ndd -set /dev/tcp tcp_ip_abort_interval 60000

/usr/sbin/ndd -set /dev/tcp tcp_conn_req_max_q 8000
/usr/sbin/ndd -set /dev/tcp tcp_conn_req_max_q0 10240
```

6.1.3 Checking ndd parameters

```
get_tcp
#! /bin/sh
export TMP TOKEN
for TOKEN in  \
tcp_time_wait_interval \
tcp_fin_wait_2_flush_interval \
tcp_conn_req_max_q \
tcp_conn_req_max_q0 \
tcp_keepalive_interval \
tcp_rexmit_interval_initial \
tcp_rexmit_interval_min \
tcp_rexmit_interval_max \
tcp_smallest_anon_port \
tcp_slow_start_initial \
tcp_xmit_hiwat \
tcp_recv_hiwat \
tcp_ip_abort_interval \
tcp_ip_abort_cinterval \
tcp_deferred_ack_interval \
ip_ignore_redirect \
tcp_conn_grace_period \
ip_path_mtu_discovery \
tcp_mss_max_ipv4 \
tcp_strong_iss
do
TMP=`/usr/sbin/ndd -get /dev/tcp $TOKEN`
echo $TOKEN=$TMP
done
for TOKEN in  \
arp_cleanup_interval
do
TMP=`/usr/sbin/ndd -get /dev/arp $TOKEN`
echo $TOKEN=$TMP
done
#EOF
```

6.2 Monitoring TCP packets

6.2.1 packets_per_second

```
#! /bin/sh
export PORT PACKETS NOW BEFORE AFTER TIME_INTERVAL
NOW=./unix_now
if [ ! -f $NOW ]
then
     echo "Perl script not found : $NOW"
     exit
fi
if [ $# -ne 2 ]
then
     echo "Usage : `basename $0` [PORT] [PACKETS]"
     exit
else
     PORT=$1
     PACKETS=$2
fi
while true
do
BEFORE=`$NOW`
snoop -o /tmp/snoop.bin -c $PACKETS dst port $PORT  > /dev/null 2>&1
AFTER=`$NOW`
TIME_INTERVAL=`expr $AFTER - $BEFORE`
if [ $TIME_INTERVAL -gt 1 ]
then
packet_count=`expr $PACKETS / $TIME_INTERVAL`
echo "Packets per second = " $packet_count
fi
done
#EOF
```

6.2.2 unix_now

```
#! /bin/perl
$ctime = time();
print $ctime, "\n";
#EOF
```

6.3 Java considerations

The Tivoli EIF probe will use the IBM Java provided by Netcool/OMNIbus unless the JAVA_HOME environment variable is set. It is recommended that the JAVA_HOME is not set to other java versions unless specific issues are seen, and are resolved through the use of another version of java.

The Tivoli EIF probe runs as two processes, one for the nco_p_tivoli_eif.jar and the other for the nco_p_nonnative probe process.

6.3.1 Non-native Probe debugging

UNIX

```
NCO_P_NONNATIVE_TRANSCRIPT=$OMNIHOME/log/tivoli_eif_NONNATIVE.log
NDE_FORCE_LOG_MODULE=$OMNIHOME/log/ tivoli_eif_NONNATIVE_FORCED.log
NDE_DEFAULT_LOG_LEVEL=debug
export NCO_P_NONNATIVE_TRANSCRIPT
export NDE_FORCE_LOG_MODULE
export NDE_DEFAULT_LOG_LEVEL
```

Windows

```
NCO_P_NONNATIVE_TRANSCRIPT=C:\TIVOLI_EIF\NONNATIVE.log
NDE_FORCE_LOG_MODULE=C:\ TIVOLI_EIF\NONNATIVE_FORCED.log
NDE_DEFAULT_LOG_LEVEL=debug
```

6.4 UNIX nice

It is possible to use UNIX nice to reset the priority of the processes being used. It depends where the bottleneck is for the system, on how best to use 'nice'. The probe run's as two processes, therefore it would be necessary to experiment with the system to determine the best settings, and processes to renice.

The following example was used for a two CPU server where there was an excessive load:

```
# Fork nonnative
exec $OMNIHOME/probes/$ARCH/nco_p_nonnative "$JAVA" $NCO_JPROBE_JAVA_FLAGS -cp $CLASSPATH
$NCO_JPROBE_JAVA_XFLAGS -DOMNIHOME="$OMNIHOME" $PROGRAM "$@" &

# Check PID's
export EIF_PID NONN_PID CHECK_EIF CHECK_NONN
EIF_PID=$$
NONN_PID=`ps -ef -o "pid ppid comm" | grep $EIF_PID | grep -v grep | grep -v  sh | awk '{print $1}' | grep -v
$EIF_PID`
echo EIF_PID=$EIF_PID
echo NONN_PID=$NONN_PID
# Set NICE for EIF Probe
renice -n 100 -p ${EIF_PID}
renice -n 50 -p ${NONN_PID}
# While loop checking for PIDS
while true
do
CHECK_EIF=`ps -p $EIF_PID -o pid=`
CHECK_NONN=`ps -p $NONN_PID -o pid=`

if [ "$CHECK_EIF" -eq "$EIF_PID" -a "$CHECK_NONN" -eq "$NONN_PID" ]
then
sleep 5
else
kill -15 $EIF_PID
kill -15 $NONN_PID
echo "Process has died ... exiting"
exit
fi
done
#EOF
```

6.5 Java based posteifmsg

The EIF JARs can be used to create a platform independent variant of the posteifmsg client. This example script shows how to leverage the NCHOME setting and directory structure. The JARs are portable to other platforms. It is recommended to use the same Java as Netcool/OMNIbus or higher.

File : posteifmsg.sh

```
#! /bin/sh
#
# Wrapper script for the EIF JARS
#
# Define EIFSDKHOME for your installation
# Requires evd.jar and log.jar
#
# Define JAVA_HOME
# Java 8 recommended
#
export EIFSDKHOME=$NCHOME/omnibus/java/jars
export CLASSPATH=$EIFSDKHOME/evd.jar:$EIFSDKHOME/log.jar

JAVA_HOME=$NCHOME/platform/linux2x86/jre_1.8.0/jre

# Check for EIFSDKHOME
if [ ! -d $EIFSDKHOME ]
then
echo "EIFSDKHOME not found : $EIFSDKHOME"
exit
fi

# Check for JAVA_HOME
if [ ! -d $JAVA_HOME ]
then
echo "JAVA_HOME not found : $JAVA_HOME"
exit
fi

# Check CLASSPATH
echo $CLASSPATH | awk -F\: '{for(i=1;i<=NF;i++)print $i}'| while read file
do
if [ ! -f $file ]
then
  echo "File not found : $file"
  EXIT_NOW=1
  exit
fi
done

# Run the classses in the given JAVA
if [ -x $JAVA_HOME/bin/java ]
then
$JAVA_HOME/bin/java -cp $CLASSPATH com.tivoli.tec.event_delivery.TECAgent SENDER "$@"
else
echo "JAVA not found in $JAVA_HOME/bin/java "
exit
fi

#EOF
```

6.5.1 Java posteifmsg Usage

```
./posteifmsg.sh -f ./posteifmsg.conf -r HARMLESS -m 'This is a test message' -n 1 \
appname=POSTEIFMSG sub_source=JAVA appcomponent=TEST hostname=localhost \
appstatus=running JavaPOSTEIFMSG EVENT
```

File : posteifmsg.conf

```
# EIF settings
BufferEvents=NO
RetryInterval=5
#BufEvtPath=/tmp/posteifmsg.cache.dat
# EIF Target port
TransportList=t1_
t1_Type=SOCKET
t1_Channels=c1_
c1_Port=9998
c1_ServerLocation=localhost
#EOF
```

Which creates and event in theTivoli EIF probe as shown:

```
ClassName: JavaPOSTEIFMSG
sub_source:        JAVA
appstatus: running
hostname:  localhost
origin:    192.168.20.20
severity:  HARMLESS
source:    EVENT
msg:       'This is a test message'
appcomponent:      TEST
EventSeqNo:        1
```