Technical Guide

Traffic Control

Feature Overview and Configuration Guide

Introduction

This guide describes **Traffic Control** and how to configure it. This new Traffic Control feature replaces the existing feature called Traffic Shaping. Traffic Control (often referred to as Quality of Service or QoS) optimizes the service provided to users when interfaces become oversubscribed.

This means creating policies that:

- identify which traffic belongs to which services
- apply different control parameters to the traffic belonging to different services

These control parameters are applied to traffic to optimize the services and can be applied in a variety of combinations:

- prioritization
- bandwidth limiting
- marking
- egress scheduling

This document explains each of the component aspects of Traffic Control including relevant commands and example configurations. The components covered are:

- Policies, see "Policies" on page 3
- Classifying traffic using rules, see "Classifying traffic using rules" on page 6
- Traffic classes and queues, see "Traffic classes and queues" on page 9
- Scheduling algorithms, see "Queueing and scheduling algorithms" on page 12
- Bandwidth limiting, see "Bandwidth limiting and burst control" on page 15

AlliedWare Plus™ OPERATING SYSTEM

Contents

Products and software version that apply to this guide

This guide applies to AR-Series Firewalls, running version **5.4.6** or later. For more information, see the following documents:

- The product's Datasheet
- The product's Command Reference

These documents are available from the above links on our website at alliedtelesis.com.

From version 5.4.7-0.1 onwards, traffic control is disabled by default on bridged traffic - see "Applying traffic control to bridging" on page 29.

Overview of Traffic Control

Data networks are notable for the immense variety of data communications that they multiplex. The primary purpose of Traffic Control is to pick out different strands of communication from this highly multiplexed river of packets and apply different treatment to different strands, or sets of strands.

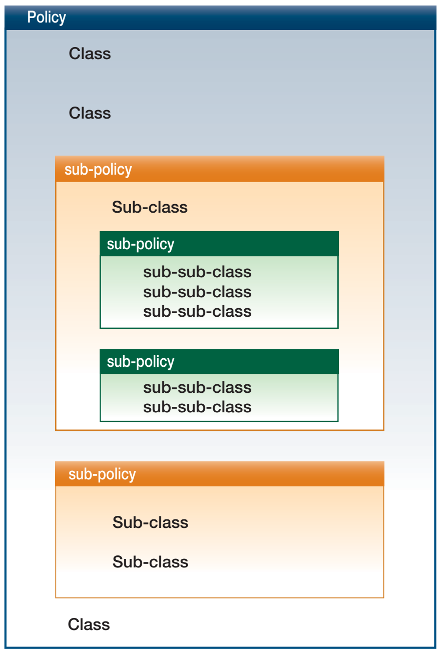
This involves setting up a series of internal forwarding paths for different traffic types, identifying which traffic type each incoming packet belongs to, and de-multiplexing the different types into their designated internal forwarding paths.

Major Components of Traffic Control

Policies

A policy defines all the control that will be applied to traffic passing between given sets of interfaces. There is a structure within the configuration of a Traffic Control policy that contains sets of traffic classes and child sub-policies.

Figure 1: Traffic Control policy structure



At the top level of the structure is the policy itself which has a name that is applied to it. The next level of the structure contains the classes - each policy can contain multiple classes. Sub-policies can sit amongst the classes.

The structure within a sub-policy is the same as that of the top level policy. There are a set of sub-classes within the sub-policy, and yet another layer of sub-policies can be interspersed among these classes.

The policy hierarchy only goes three deep, so the policies at the third level of depth contain only classes, and no further level of sub-policies.

For more information about the relationship between classes, sub-policies, sub-classes and sub-sub-classes see "Policy hierarchy" on page 10.

Traffic classification

Classification is the process of examining packets to decide which traffic category they belong to. This is achieved by creating a set of rules that define the different traffic categories, and then applying those rules to each incoming packet, to allocate it to one of the categories.

The process of demultiplexing all the communication strands so that different polices can be applied to different strands is complex. There are a wide range of different factors that can be used to differentiate between different traffic categories.

However, the classification rules that are applied to the demultiplexing process must be comprehensible to users, as it is users that write the rules, in the form of configuration commands on the Next Generation Firewall device.

Therefore, AlliedWarePlus defines two concepts that assist with making the classification process comprehensible, and able to be configured in a tractable manner.

The two concepts are: **Applications** and **Entities**.

Applications are quite a straightforward concept. Simply, these are the end-user application that is generating a data flow. A full list of applications can be displayed using the **show application** or **show application detail** command.

Examples of applications include:

- FTP file transfer
- Skype voice call
- Email
- Web Browsing
- Facebook

The Next Generation Firewalls have a sophisticated deep-packet inspection engine that uses a range of rules to determine the application that generated any given packet. The rules can be as simple as “TCP destination port 25 = SMTP email”, but in general they involve searching deep into the packet’s data payload to find byte patterns that are characteristic of different applications.

An **Entity** is an overall term for a network or network element that identifies where a packet has come from or where it is going to.

AlliedWare Plus™ defines three levels of Entity:

- **Host**—the most granular level of entity is an individual network host. A host will be represented by its IP address.
- **Network**—the next level of granularity is a subnet (or collection of subnets) and the network interface via which it is accessed.
- **Zone**—a high level abstraction for a logical grouping of networks. The minimum zones normally implemented would be a trusted zone for the private network behind the firewall and an untrusted zone for the Internet. Other common zones are a Demilitarised Zone (DMZ) for publicly visible web servers and a Virtual Private Network zone for remote access users or tunnels to other networks.

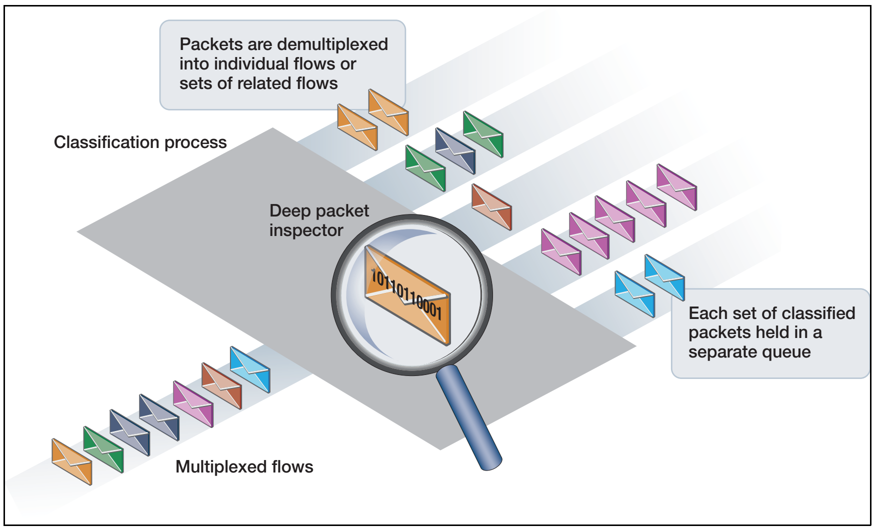
AlliedWare Plus designates a data flow to be a series of packets, generated by an identified application, flowing from an identified source Entity towards an identified destination Entity.

For example, an:

- FTP file transfer from the Trusted Private Zone to the Untrusted Internet Zone.
- Email from the Untrusted Internet Zone to a specific email server host in the DMZ.
- RDP (Remote Desktop) connection from the VPN zone into the Trusted Private Zone.

The classification process, therefore, is the process of classifying packets into these types of data flows. The user creates classification rules (we will look at the syntax of these rules further below) to instruct the Traffic Control policy on which flows to look for. All packets that do not match any of the designated flows simply fall into the 'everything else' category.

Figure 2: Classification of data process


Packets are demultiplexed into individual flows or sets of related flows
Classification process
Deep packet inspector
10110110001
Each set of classified packets held in a separate queue
Multiplexed flows

Classifying traffic using rules

Rules are the mechanism for classifying traffic and determining what type of controls the AR-Series Firewall applies to the different classifications. A Traffic Control configuration can contain multiple rules, which the firewall traverses in numerical order to find a match for a given data stream, and thereby decide the controls that are to be applied to that stream.The process of traffic classification consists of matching traffic against rules.

Match criteria

A rule matches on the following three criteria:

1. **What application does the traffic belong to?**

   The AR-Series Firewall Deep Packet Inspection engine determines each stream's application. The device can recognize thousands of applications that are defined by a combination of:

   a. the pre-defined application pattern list

   b. the application signatures provided by the Procera subscription service

   c. application definitions that can be created in Application config mode (entered by using the **application** command)

2. **Where did the traffic come from?**

   The direction from which traffic originates is expressed as an Entity: Zone, Network, or Host.

   Some rules may apply to all traffic that comes from a set of subnets, in which case, the 'From Entity' for the rule would be the Zone that encompasses those subnets. Other rules may apply to traffic that is just from an individual host, in which case the 'From Entity' for the rule will be a 'Host Entity'.

3. **Where is the traffic going to?**

   Similar to the origin direction of traffic, its destination is also expressed as an Entity. Just as with the 'From Entity', the 'To Entity' on a rule can be a Zone, a Subnet, or a Host.

   The 'To Entity' of a rule serves not only as one of the parameters that classifies which traffic matches the rule, it also identifies the egress interface(s) to which the rule applies.

Because traffic control is actually applied at the **egress** interface, the identification of the egress interface or interfaces for a rule is significant. It determines the interface to which the rule's policy is attached, and therefore on which interface the policy's classes will queue traffic.

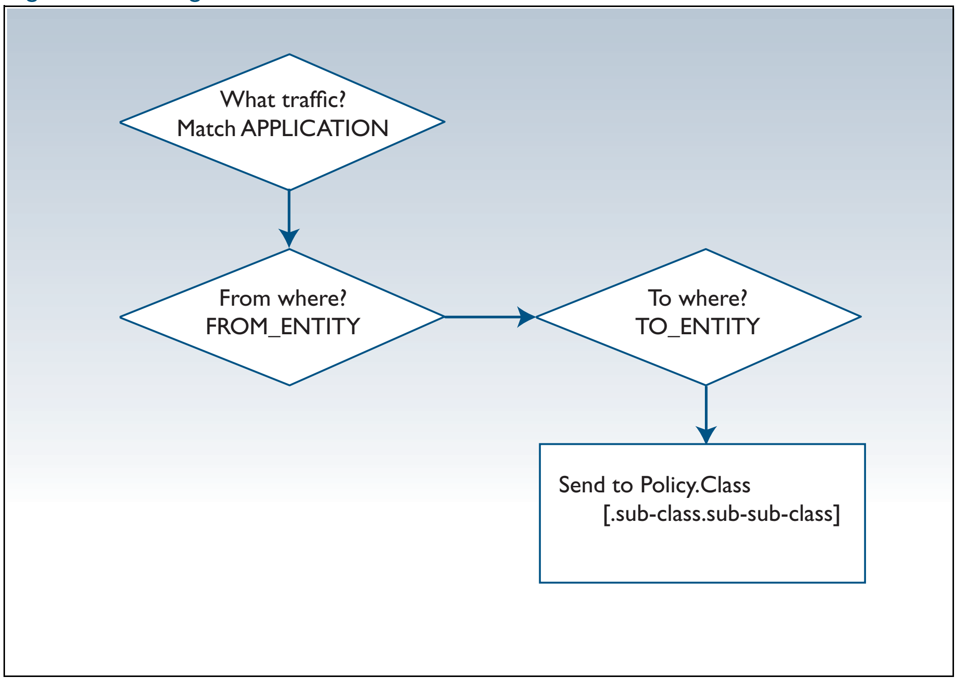
Assign a class to a policy

The fourth parameter in a rule is the traffic class to which the matching traffic will be assigned. The identity of the class is expressed as a **Class Name** that belongs to a given policy. The identifier depends on where the assigned class sits in the policy hierarchy, i.e. if it is a class within the top-level policy or one within a sub-level policy or a sub-sub-level policy. So the identifier is written as one of the following:

- policy_name.Class_name
- policy_name.Class_name.sub-sub-class_name
- policy_name.Class_name.sub-class_name.sub-sub-class_name

The logic of rule execution is explained in the following figure.

Figure 3: The logic of rule execution



Configuring a rule

There are various constraints and freedoms when configuring rules:

- You do not have to configure the rule-ID parameter when defining a rule. If it is not configured, then the rule is automatically assigned an ID equal to the next exact multiple of 10 that is larger than the current highest-numbered rule. For example, if the current highest-numbered rule is rule 56, and you create a rule without a specific rule-ID, it will be automatically assigned the ID 60.
- You cannot give a rule the same rule-ID as an existing command. An attempt to do so will elicit an error message.
- You do not have to specify an application. If it is not specified, then that rule will match all types of traffic.
- If you do specify an application, and that application does not exist in the set of applications known by the device, then the rule will be accepted, but will not ever be executed because no traffic will ever be deemed to match this unknown application.
- You do not have to specify the 'From Entity' on a rule. If it is not specified, then the rule matches traffic arriving from any source.
- You do have to specify the 'To Entity' on a rule, as this is the parameter that defines the egress interface(s) on which the traffic matching the rule will be controlled.
- You do have to specify a class on a rule. This class has to be a leaf class. For more information, see "Sub policy classes and leaf classes" on page 12.

Syntax for creating a rule

```
awplus(config-tc)#rule <rule-ID> match <application>
from <source-entity> to <dest-entity>
policy <policy-name.class-name[.subclass-name[.subsubclass.name>]]
```

Changing a rule

It may be necessary to change the ID of a rule, in which case you can use the **move rule** command.

- rule-ID is the existing rule to change in the range from 1 to 65535
- new-ID is the new rule number to assign to the existing rule in the range from 1 to 65535. This number cannot be the ID of an existing rule.

Syntax for moving a rule

```
awplus(config-tc)#move <rule-ID> to <new-ID>
```

Traffic classes and queues

For each packet that is passed to it, Traffic Control will either:

- drop the packet, or
- place the packet into one of the queues that hold packets waiting to egress an interface.

The purpose of traffic classification rules is to allocate packets to traffic classes. Each of these classes represents a queue that holds packets waiting to egress an interface. The classes have criteria for deciding whether a given packet that is assigned to them gets put into the queue. There are two reasons why a packet will be dropped rather than be put into the queue:

1. It is outside the bandwidth limits for the class to which it is assigned. In other words, the class is limited to sending x bits per second, and has recently been sending packets at such a rate that it has reached its limit and must pause a while before accepting any more packets to send.
2. The queue belonging to the class to which the packet is assigned is full.

Queues on egress interfaces

Because traffic control is implemented in software, there is great flexibility in the number and nature of the queues that can exist on a given egress interface. This is in contrast to the case of a hardware switch, which will typically have a fixed set of 4 or 8 queues per port, and relatively few configurable properties on those queues.

By default, an egress interface on the AR-Series Firewalls has just a single vanilla queue. But as the traffic control configuration is executed, new queues are dynamically created on interfaces. The trigger for creating a queue on a given egress interface is the configuring of a traffic control rule where:

- The ‘To Entity’ in the rule includes the interface (i.e. the interface in question is one via which the ‘To Entity’ can be reached).
- The class in the rule has not been combined with the same ‘To Entity’ in any existing rule. If the same class and ‘To Entity’ combination had already been specified in an existing rule, then the queue in question would already have been created.

When the rule is configured, identical queues are created on all the egress interfaces that are covered by the ‘To Entity’ (i.e. all the interfaces via which the ‘To Entity’ is reached).

Queues are associated with egress interfaces, as the purpose of the queues is to corral the packets in preparation for transmitting them out through an egress interface.

The properties of the queue (its length, its bandwidth limits, its priority etc.) are defined by the properties configured on the class in question.

Class characteristics

When many different types of traffic are competing for bandwidth, it is desirable to have a high degree of control over the allocation of that bandwidth. For example, a network administrator may desire to control traffic in the following manner:

- One particular type of traffic is given top priority treatment, so that packets of this type always get through, with minimum delay.
- A set of three or four traffic types sit at the next priority, operating in a round robin fashion, each with a maximum bandwidth limit.
- A set of four further traffic types at the next priority level, equally sharing a specified amount of bandwidth.

A scenario of this nature requires some complexity in the structure of the policies that represent the way traffic is classified, and how the supply of egress bandwidth is allocated to different classes of traffic. Therefore the traffic control implementation on the AR-Series Firewalls features:

- A three-level hierarchy of policies
- Three different algorithms for scheduling the transmission of packets from queues
- A range of bandwidth limiting and burst control parameters

Policy hierarchy

To enable scenarios like that described above, where the different types of traffic are treated as separate sets with certain controls applied within each set, and other methods used to control the sets relative to each other - there needs to be a hierarchy of traffic control policies.

In practice, this means that some or all of the classes within a policy can be sub-policies, and can have their own set of child sub-classes. Some or all of these sub-classes can be sub-sub policies, with their own sets of child sub-sub-classes.

For example, a class within a set of priority classes can be manifested by a sub-policy that uses a different algorithm, and contains its own classes and sub-sub-policy.

Example syntax

```
class A priority-level 15
class B priority-level 10 sub-class-policy <algorithm>

 sub-class GOLD <parameters...>
 sub-class SILVER <parameters...> sub-sub-class-policy <algorithm>
  sub-sub-class MED <parameters...>
  sub-sub-class LOW <parameters...>
```

In this example, the AR-Series Firewall processes traffic as follows:

1. Traffic in class A is transmitted, because it has a higher priority
2. If there is no traffic in class A, traffic in class B is processed:

    a. traffic in sub-class GOLD is processed according to its sub-class parameters

    b. traffic in sub-class SILVER is processed according to the parameters of sub-sub-classes MED and LOW. Note that traffic in sub-sub-classes MED and LOW is constrained overall by the parameters on sub-sub-class policy SILVER (e.g. the total bandwidth assigned to that policy and the burst levels allowed for that policy).

The way Traffic Control views classes and policies needs to be clarified at this point. Entities B and SILVER can be considered as both classes and policies.

As far as the parent policy is concerned, it sees just a set of classes within it (Entities which have packets to deliver, and which are competing for the opportunity to deliver those packets). The parent policy makes no distinction between simple classes and those that are actually sub-policies. Therefore, as far as the sub-policy B is concerned, the entity SILVER appears as a class, just like GOLD. From the top looking down, these entities appear as classes.

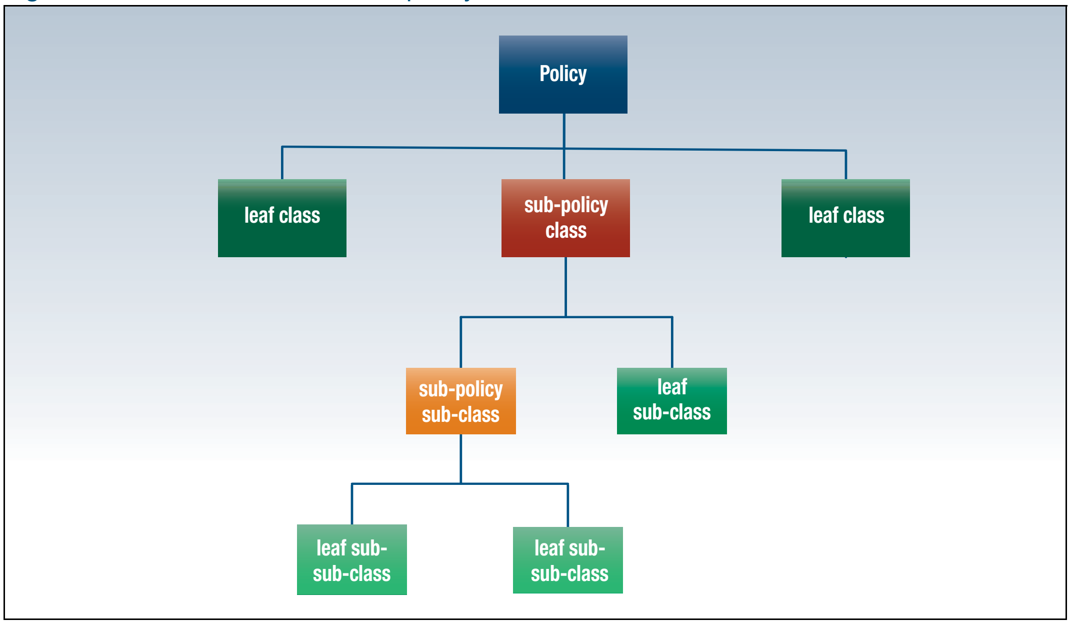
However, as far as the child classes within a sub-policy are concerned, the Entities like B and SILVER are policies – they define the bandwidth resources for which those classes compete, and define the process for deciding which class gets to transmit the next packet. So, from the bottom looking up, these Entities are policies.

Child policies have to operate within the context of the higher level policies above them. That is, the higher level policies control the bandwidth available to the sub-policies and the opportunities they have for transmitting any packets. Within these constraints, the classes within the child policies operate independently of classes in other child policies.

Sub policy classes and leaf classes

The section above describes how classes can also act as sub-policies. There are, of course, classes that do not operate as sub-policies. These classes are referred to as “leaf classes”, because they are at the end of a branch—there are no further sub-classes extending from them, see the following figure.

Figure 4: Hierarchical view of sub policy classes and leaf classes



Leaf classes are the classes to which rules assign traffic, and that map to queues on egress interfaces. The sub-policy classes never have traffic classified to them, instead, they provide the overall parameters within which their child classes must operate. This is why the classes configured in **rules** must be leaf classes.

Queueing and scheduling algorithms

The purpose of QoS is to control how different types of traffic get access to the bandwidth available on congested egress interfaces.

There are two points at which control can be applied:

1. The queueing decision as to whether a given packet gets into an egress queue, or is dropped
2. The scheduling decision as to which egress queue is allowed to transmit the next packet

Traffic control on the AR-Series Firewalls defines three types of policies, each of which uses the same queueing algorithm but has a different scheduling algorithm. The queueing algorithm they all use is Stochastic Fairness Queueing (SFQ).

The three types of policies are:

1. **Strict Priority Policies**

   Scheduling algorithm - Strict Priority
   The classes each have a priority value. The class with the highest priority gets to send packets first. Only when it has nothing to send does the next priority queue get to send any packets, and so on.

2. **Weighted Round Robin (WRR)**

   Scheduling algorithm - Weighted Round Robin
   The weighted round robin algorithm is useful for simple configurations where it is desired that different classes each get guaranteed at least a certain percentage of the available bandwidth, but fine grained control is not needed. It works by assigning a weight to each class, and then services each class in turn, allowing each class to send packets according to its weight. If there are no packets queued for a particular class, other classes can send more traffic as their turn to send will come around faster. The amount of data to be sent is based on the size rather than the number of packets.

3. **Hierarchical Token Bucket (HTB)**

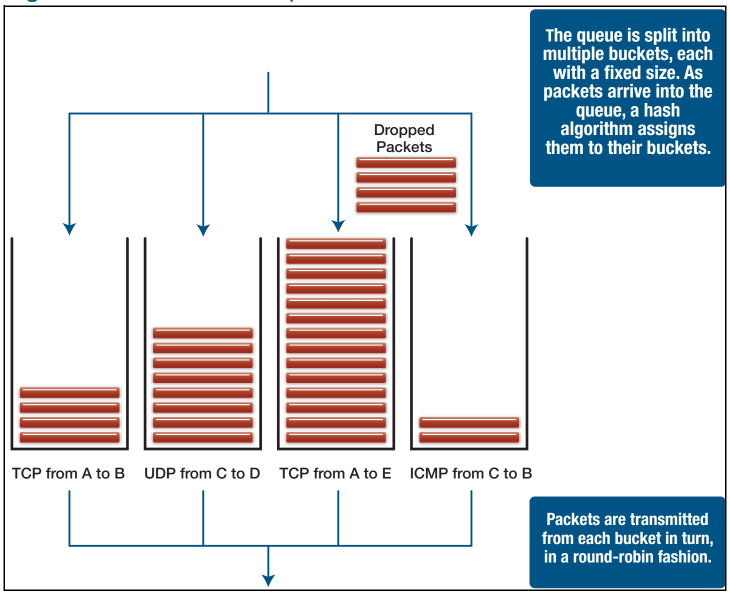
   Scheduling Algorithm - Strict Priority
   Each class is configured with a preference value. Classes with a higher preference value are able to transmit packets first, and have higher priority when being given excess bandwidth. For more information about defining the characteristic of HTB policies and their bandwidth limiting scheme, see "Bandwidth limiting and burst control" on page 15.

SFQ queueing algorithm

As mentioned above, all the policy types use SFQ as the queueing algorithm. The queue is divided into a series of hash buckets. Each packet is dropped into a particular bucket, based on a hash calculation performed on its stream parameters (addresses and TCP/UDP ports). If the number of packets in the queue exceeds the configured queue length, packets will be tail-dropped from the fullest bucket.

When a packet is to be transmitted from the queue, it is not simply the packet at the head of the queue that is transmitted. Instead, the algorithms transmit packets from different buckets in a round-robin manner.

Figure 5: Traffic Control packet stream


The queue is split into multiple buckets, each with a fixed size. As packets arrive into the queue, a hash algorithm assigns them to their buckets.
Dropped Packets
TCP from A to B
UDP from C to D
TCP from A to E
ICMP from C to B
Packets are transmitted from each bucket in turn, in a round-robin fashion.

The goal of this algorithm is to give fair treatment to all the different streams being managed by the class. The system would be very fair if there was only one stream per bucket, but actually multiple streams can hash to the same bucket, so if one stream is being very greedy it will cause unfairness to the other streams that hash to the same bucket. For this reason, the hash algorithm is changed at regular intervals, so that the sets of streams that share buckets change regularly. This way, the streams that have to suffer the fate of sharing buckets with greedy streams are regularly changed, so no streams are stuck with that fate for too long.

Bandwidth limiting and burst control

Each policy type has its own approach to controlling the bandwidth that each class within it can use.

- Strict Priority policies have an optional maximum bandwidth that can be configured on classes. This is a simple limit that just shapes the maximum rate at which packets can be transmitted from the class.
- WRR policies do not have an ability to bandwidth limit individual classes. This is because the very nature of WRR is that the weights effectively allocate each a percentage of the available bandwidth to each class.
- HTB policies are characterised by the fairly complex bandwidth limiting and burst control features that the HTB algorithm provides. HTB enables the user to assign bandwidth limits to individual classes. HTB classes can be configured using the following parameters:
  - Committed Information Rate (CIR)
    This is the guaranteed bandwidth of traffic in this class.
  - Peak Information Rate (PIR)
    This is the maximum rate at which this class can send if the parent class has spare bandwidth. This defaults to the CIR, in which case no bandwidth is borrowed.
  - Committed Burst size (BC)
    This is the number of bytes that can be burst at rates higher than the CIR. This should be at least as high as that of any child class.
  - Excess Burst size (BE)
    This is the number of bytes that can be burst at rates higher than the PIR. This should be at least as high as that of any child class.

Restrictions on the types of child policies

Strict Priority policies have no restriction on the types of policies that can be created within them. Any policy type can be nested within a priority policy.

However:

- only HTB policies can be nested inside an HTB policy
  and
- only WRR policies can be nested inside WRR policies

Configuring policies and classes

To configure the root policy, use the commands:

Root policy

```
awplus#configure terminal
awplus(config)#traffic-control
awplus(config-tc)#policy <policy-name> {priority|wrr|htb}
```

The priority algorithm specified on the command defines the algorithm that is used for scheduling between the classes in the top level policy.

To configure a leaf class in a Strict Priority policy, use the commands:

Strict Priority class

```
awplus#configure terminal
awplus(config)#traffic-control
awplus(config-tc)#policy <policy-name> priority
awplus(config-tc-policy)#class <class-name> priority-level <0-15>
[max <max-rate>] [queue-length <2-65536>] [set-dscp <dscp>] [red-
curve <red-curve-name>]
```

similarly, for a sub-class as a leaf, use the commands:

```
awplus#configure terminal
awplus(config)#traffic-control
awplus(config-tc)#policy <policy-name> priority
awplus(config-tc-policy)#class <class-name> priority-level <0-15>
sub-class-policy priority
awplus(config-tc-class)#sub-class <sub-class-name> priority-level
<0-15> [max <max-rate>] [queue-length <2-65536>] [set-dscp <dscp>]
[red-curve <red-curve-name>]
```

and, for a sub-sub-class as a leaf, use the commands:

```
awplus#configure terminal
awplus(config)#traffic-control
awplus(config-tc)#policy <policy-name> priority
awplus(config-tc-policy)#class <class-name> priority-level <0-15>
sub-class-policy priority
awplus(config-tc-class)#sub-class <sub-class-name> priority-level
<0-15> sub-sub-class-policy priority
awplus(config-tc-subclass)#sub-sub-class <sub-sub-class-name>
priority-level <0-15> [max <max-rate>] [queue-length <2-65536>]
[set-dscp <dscp>] [red-curve <red-curve-name>]
```

To configure a leaf class in a WRR policy, use the commands:

WRR class

```
awplus#configure terminal
awplus(config)#traffic-control
awplus(config-tc)#policy <policy-name> wrr
awplus(config-tc-policy)#class <class-name> weight <1-100> [queue-length
<2-65536>] [set-dscp <dscp>] [red-curve <red-curve-name>]
```

similarly for a sub-class as a leaf, use the commands:

```
awplus#configure terminal
awplus(config)#traffic-control
awplus(config-tc)#policy <policy-name> wrr
awplus(config-tc-policy)#class <class-name> weight <1-100> sub-class-
policy wrr
awplus(config-tc-class)##sub-class <sub-class-name> weight <1-100>
[queue-length <2-65536>] [set-dscp <dscp>] [red-curve <red-curve-name>]
```

and for a sub-sub-class as a leaf, use the commands:

```
awplus#configure terminal
awplus(config)#traffic-control
awplus(config-tc)#policy <policy-name> wrr
awplus(config-tc-policy)#class <class-name> weight <1-100> sub-class-
policy wrr
awplus(config-tc-class)#sub-class <sub-class-name> weight <1-100>
sub-sub-class policy wrr
awplus(config-tc-subclass)#sub-sub-class <sub-sub-class-name> weight
<1-100> [queue-length <2-65536>] [set-dscp <dscp>] [red-curve <red-
curve-name>]
```

Note: The sub policies within a WRR policy must also use the WRR algorithm.

To configure a leaf class in a HTB policy, use the commands:

HTB class

```
awplus#configure terminal

awplus(config)#traffic-control

awplus(config-tc)#policy <policy-name> htb

awplus(config-tc-policy)#class <class-name> cir <committed-rate>
[pir <peak-rate>] [bc <burst>] [be <excess-burst>] [preference
<0-7>] [queue-length <2-65536>] [set-dscp <dscp>] [red-curve <red-
curve-name>]
```

similarly for a sub-class as a leaf, use the commands:

```
awplus#configure terminal

awplus(config)#traffic-control

awplus(config-tc)#policy <policy-name> htb

awplus(config-tc-policy)#class <class-name> cir <committed-rate>
[pir <peak-rate>] [bc <burst>] [be <excess-burst>] [preference
<0-7>] sub-class-policy htb

awplus(config-tc-class)#sub-class <sub-class-name> cir <committed-
rate> [pir <peak-rate>] [bc <burst>] [be <excess-burst>]
[preference <0-7>] [queue-length <2-65536>] [set-dscp <dscp>] [red-
curve <red-curve-name>]
```

and for a sub-sub-class as a leaf, use the commands:

```
awplus#configure terminal

awplus(config)#traffic-control

awplus(config-tc)#policy <policy-name> htb

awplus(config-tc-policy)#class <class-name> cir <committed-rate>
[pir <peak-rate>] [bc <burst>] [be <excess-burst>] [preference
<0-7>] sub-class-policy htb

awplus(config-tc-class)#sub-class <sub-class-name> cir <committed-
rate> [pir <peak-rate>] [bc <burst>] [be <excess-burst>]
[preference <0-7>] sub-sub-class-policy htb

awplus(config-tc-subclass)#sub-sub-class <sub-sub-class-name> cir
<committed-rate> [pir <peak-rate>] [bc <burst>] [be <excess-burst>]
[preference <0-7>] [queue-length <2-65536>] [set-dscp <dscp>] [red-
curve <red-curve-name>]
```

DSCP marking

One of the actions that a class can apply to packets is to set a value in the DSCP field of any packets it transmits. This gives downstream devices information about the priority that the AR-Series Firewall applied to the packet stream.

Queue lengths

The default queue length for every leaf class is 1000. Each SFQ bucket with each queue can hold up to 127 packets. The bucket size is not configurable but the overall length of any queue is configurable.

Other Features of Traffic Control

Virtual bandwidth

For various reasons, you may wish to limit the total bandwidth that can be transmitted from an interface. You may wish to constrain an interface to transmit at only 50Mbps, even though it is linked up at 100Mbps. This artificial reduction in an interface's bandwidth is achieved by configuring a virtual bandwidth on the interface. The virtual bandwidth is the maximum rate of traffic that traffic control will allow through an interface.

We recommend you set the virtual bandwidth for all interfaces used for traffic control to ensure that traffic is actually rate limited as expected. Most interfaces have a default bandwidth of 1Gbps for control purposes.

To set a virtual bandwidth of 900Mbps for vlan3, use the commands:

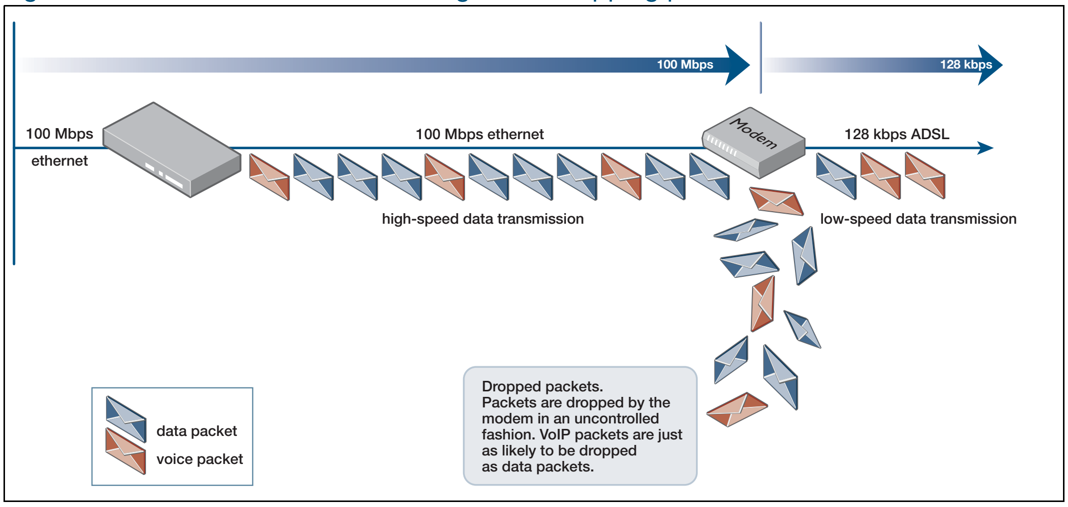
```
awplus#configure terminal
awplus(config)#traffic-control
awplus(config-tc)#interface vlan3 virtual-bandwidth 900mbit
```

The most common usage of virtual bandwidth is to:

1. Associate a realistic bandwidth with a tunnel interface, which does not have an inherent bandwidth.
2. Deal with the situation where a device with little or no QoS capability sits between the AR-Series Firewall and a lower-bandwidth data link.

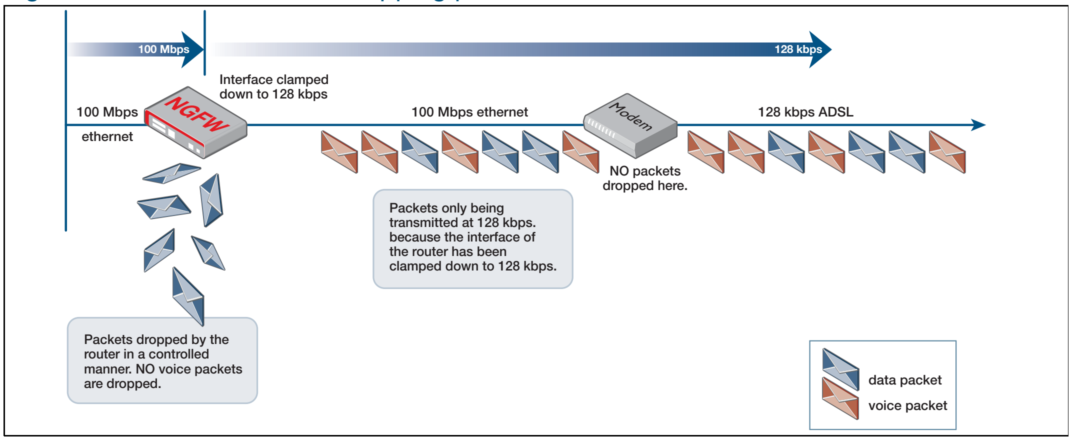
The most common such circumstance is the case where the AR-Series Firewall is connected by Ethernet to a modem or low-end router, which in turn is connected to a lower-bandwidth WAN link. If the data rate being sent out the Ethernet interface toward the modem/router exceeds the capacity of the WAN link, then the modem/router will drop packets. If that modem/router device has little traffic management capability, then the packets will just be dropped in an uncontrolled manner.

Figure 6: Device with little traffic management dropping packets in uncontrolled manner


100 Mbps
128 kbps
100 Mbps ethernet
100 Mbps ethernet
Modem
128 kbps ADSL
high-speed data transmission
low-speed data transmission
data packet
voice packet
Dropped packets. Packets are dropped by the modem in an uncontrolled fashion. VoIP packets are just as likely to be dropped as data packets.

However, if the egress bandwidth of the AR-Series Firewall's Ethernet interface is artificially limited to the bandwidth of the downstream WAN link, then the packet dropping will be performed, in a controlled manner, in the AR-Series Firewall.

Figure 7: AR-Series Firewall dropping packets in a controlled manner


100 Mbps
128 kbps
Interface clamped down to 128 kbps
100 Mbps ethernet
NGFW
100 Mbps ethernet
Modem
128 kbps ADSL
NO packets dropped here.
Packets only being transmitted at 128 kbps. because the interface of the router has been clamped down to 128 kbps.
Packets dropped by the router in a controlled manner. NO voice packets are dropped.
data packet
voice packet

System class

A special policy called the system class is automatically provided on all interfaces. Packets assigned to this policy have high priority and are reserved bandwidth on an interface. Some packet types are assigned to the system class by default. These are ARP packets, IPv6 Neighbor Discovery packets, and PPP control packets.

To configure a rule to assign other traffic to the system class:

```
awplus#configure terminal
awplus(config)#traffic-control
awplus(config-tc)#rule 110 match app to A policy system
```

Note: Packets matching such a rule will be sent to the system class.

By default, 5% of the (virtual) bandwidth of interfaces used for traffic control is reserved for the system class. This percentage can be changed if desired by using the **system-bandwidth** parameter. If the bandwidth is unused, then user traffic can take advantage of this. However, it is recommended to take this into account when configuring maximum rates.

Note: There is certain system traffic that bypasses the traffic control layer and is sent directly to the hardware.

To reduce the bandwidth reserved for system traffic to 1% on eth2, use the commands:

```
awplus#configure terminal
awplus(config)#traffic-control
awplus(config-tc)#interface eth2 system-bandwidth 1
```

Overhead

The **overhead** configuration parameter allows Traffic Control to take into account overheads of sending the packet to make bandwidth calculations more accurate. It can be configured as a number of bytes to add to the calculation of the size of each packet or as the keyword **ethernet**. The keyword **ethernet** will add an overhead of 24 bytes to each packet to take into account the 12 bytes of inter-frame gap, the 8 bytes of preamble and the 4 byte CRC. This means that a rate limit of 10Mbps on the bandwidth will result in the traffic being accurately limited to the maximum line rate of a 10Mbps interface.

To configure the use of Layer 1 Ethernet framing overhead for rate calculations on interface eth2, use the commands:

```
awplus#configure terminal
awplus(config)#traffic-control
awplus(config-tc)#interface eth2 overhead ethernet
```

It is also possible to configure the overhead for other interface types such as tunnels. If the packet overhead is known or calculated, this can be added to limit the encapsulated tunnel traffic to a certain rate.

Note: Traffic Control does not allow for packets being fragmented when going into the tunnel. The overhead is only applied once per packet entering the tunnel, not once per fragment.

To apply 56 bytes of encapsulation overhead to packets for tunnel rate calculation on tunnel1:

```
awplus(config)#traffic-control
awplus(config-tc)#interface tunnel0 overhead 56
```

Default class

Every policy has an implicit default class. All traffic that does not match any of the classifier rules is assigned to this default class, which simply transmits traffic on a best effort basis, using whatever bandwidth is not being used by other classes.

This class differs from other classes in that it uses a slightly different queueing algorithm called **FQ-Code**. This is a variation on SFQ. This algorithm, rather than tail-dropping packets from the full buckets, calculates how long packets spend sitting in each bucket. When it detects that packets have been waiting too long in a bucket, it starts dropping packets from that bucket. Once the waiting times shorten, it stops dropping packets. In this way, it is able to reduce delays and induce TCP streams to adjust their sending rate to conform to the available bandwidth.

RED (Random Early Discard) curves

The goal of using RED curves in a traffic-control configuration is to cause TCP flows to back off early by occasionally dropping a packet, rather than causing all flows to back off at the same time when the queue overflows.

A RED curve defines packet dropping behaviors that depend on the length of a queue. Typically, the behavior is:

- When the queue is short, no packets are dropped.
- As the queue grows, the probability of a packet being dropped is increased.
- When the queue reaches a particular length, the probability of dropping a packet stops increasing.

In the AR-Series Firewall, RED curves are applied separately to each of the SFQ hash buckets within a traffic queue, see "SFQ queueing algorithm" on page 14 for a description of the hash buckets. This means that the control of TCP sessions can be performed on a more granular basis.

RED curves are highly configurable - the thresholds for starting to increase the drop probability, and stopping the increase probability, are configurable, as is the max drop probability at the upper threshold, and a queue size at which ALL arriving packets will be dropped.

Configuration of RED curves

There are two steps to configuring RED curves.

1. Named RED curve templates are created. These define the parameters for a RED curve.
2. A named template is applied to one or more leaf traffic classes.

The command for defining a RED curve template, entered in Traffic Control configuration mode, is:

```
awplus(config-tc)#red-curve <name> [limit <4-127>]
[avpkt <64-1518>] [min <3-126>] [max <4-127>]]
[probability <0-100>] [ecn [ecn-drop]]
```

Table 1: RED curve template parameters

PARAMETER	MEANING	DEFAULT
<name>	Name of RED curve template	
limit <4-127>	Size of real queue at which all packets in the flow (hash-bucket) will be dropped. (This value multiplied by average packet size). This value is limited to 127 because the maximum queue depth for a single flow is 127.	127
avpkt <64-1518>	Average packet size to use for calculations.	576
min <3-126>	Multiples of avpkt size at which to start increasing drop probability (based on calculated average queue size).	10

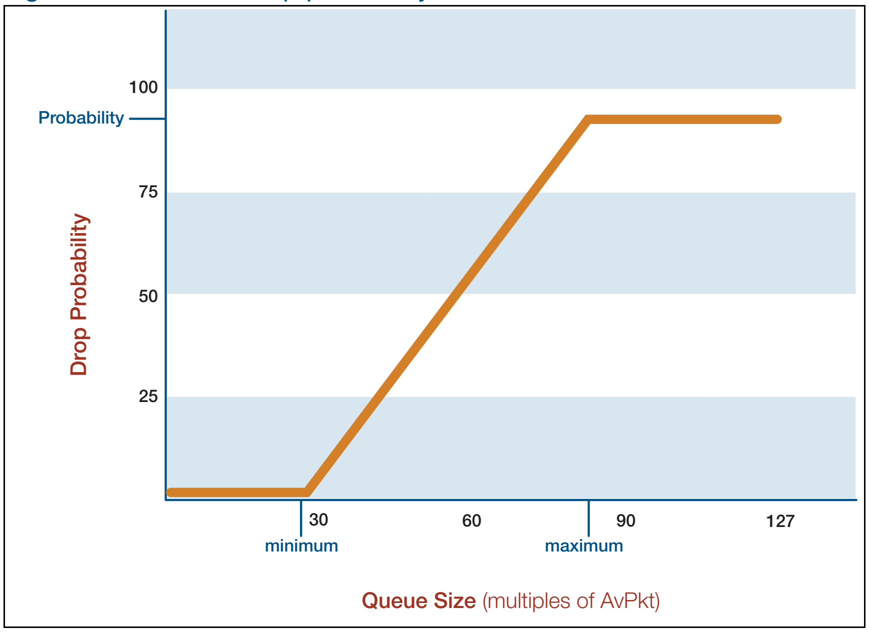
Table 1: RED curve template parameters

PARAMETER	MEANING	DEFAULT
max <4-127>	Multiples of avpkt size at which maximum drop probability is reached. If the average queue size exceeds this, all packets will be dropped or ECN marked.	32
Probability <1-100>	Maximum drop probability as a percentage	2
ecn	ECN mark packets that are marked as being ECN capable instead of dropping.	No
ecn-drop	If <max> is exceeded and ECN marking is configured, drop packets instead of marking.	No

Note: All the parameters in the command for defining the RED curve template have default values. If a given parameter is not specified in the command, the default value of the parameter is used.

The diagram below illustrates the relationship between the values min, max, and probability. The resulting graph of drop probability against the level of data in the hash bucket is the standard RED curve profile.

Figure 8: Red curve drop probability

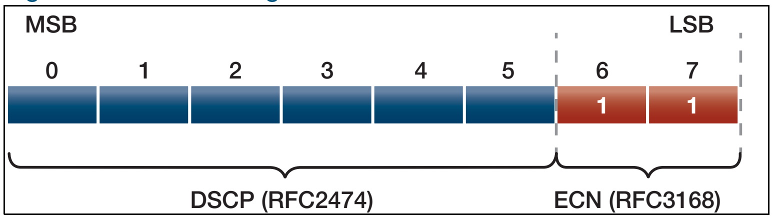


The command requires that the value of **min** cannot be greater than that of **max**. If the command incorrectly specifies a min value that is greater than the max value, the follow error message results:

```
% max(X) must be greater than min(Y)
```

The ECN marking that is applied if the ECN parameter is configured, is that the two least significant bits of the Diffserv field of IPv4 or IPv6 packets are both set to 1, which indicates “Congestion Encountered” (CE). For a more detailed description about the operation of ECN with IP, see the article provided on Wikipedia.
(https://en.wikipedia.org/wiki Explicit_Congestion_Notification#Operation_of_ECN_with_IP)

Figure 9: ECN marking



The purpose of the **avpkt** parameter is to convert the limit value from packets to bytes. The underlying module that implements the RED curve functionality performs its calculations in terms of bytes - i.e. its threshold for beginning to drop packets is actually triggered when a certain number of bytes are stored in the hash bucket, rather than when a certain number of packets are accumulated. This is because accumulated bytes are truer indication of congestion than accumulated packets (a queue several small packets may constitute a small amount of queued data, whereas a queue of a few large packets may constitute a large amount of queued data). So, the **avpkt** parameter gives a rule for deciding how many bytes the value of limit (defined in packets) represents; and therefore how many bytes the min and max values represent.

RED curves are applied to Leaf Traffic classes with the command (in Traffic-Class configuration mode):

```
class <name> red-curve <name>
```

If the RED curve template specified in the command does not exist, an error message is output, and the command is not executed. RED curve template names are case sensitive.

When a RED curve template has been applied to a traffic class, the parameters defined in the RED curve template are replicated across all the SFQ hash buckets within the egress queue(s) which hold the traffic belonging to that traffic class.

The command that defines a RED curve template can also be used to update the parameters of an existing RED curve template. If the parameters on a RED curve template are altered, then the alteration flows through to all the Traffic classes to which that template has been applied, and therefore to all the Hash buckets within those traffic classes' queues.

Show commands

The details of all currently configured RED curve templates can be seen in the output of the command: **show traffic-control red-curve**.

```
awplus#show traffic-control red-curve
Traffic Control RED Curves:
RED curve default:
  Limit:                 127 packets
  Average packet size:   576B
  Minimum:               8%
  Maximum:               25%
  Drop probability:      2%
  ECN marking:           disabled

RED curve R1:
  Limit:                 3 packets
  Average packet size:   576B
  Minimum:               8%
  Maximum:               25%
  Drop probability:      2%
  ECN marking:           disabled


RED curve R2:
  Limit:                 20 packets
  Average packet size:   576B
  Minimum:               10%
  Maximum:               30%
  Drop probability:      2%
  ECN marking:           disabled
```

Or, details of a single RED-curve template can be displayed:

```
awplus#show traffic-control red-curve R1

RED curve R1:
  Limit:                 3 packets
  Average packet size:   576B
  Minimum:               8%
  Maximum:               25%
  Drop probability:      2%
  ECN marking:           disabled
```

Performance considerations

There is a good deal of overhead involved in processing packets in the traffic control module, so it makes sense not to process every packet that comes along, if possible.

The way that traffic control works in the AR-Series Firewalls is that the first packet of any new stream is given a full processing, and a classification is applied to it. The result of that classification is cached, so that any further packets in the same stream will bypass all the rule look-up process and will automatically be given the same classification as the first packet.

This improves the performance of traffic control immensely, but does have a couple of effects to be aware of:

1. If the configuration of traffic control rules are altered, then the rule changes will not be applied to any existing streams, they will continue to use their cached classification. All new streams, of course, will be processed according to the new rule configuration.
2. If any traffic is being classified by an application that includes DSCP as one of the criteria that defines the application, then all packets in a stream whose first packet matches this application will receive the same classification. If some of the packets in the stream do not have the same DSCP value as the initial packet of the stream, they will be treated as if they did.

Configuration Examples

The steps to configuring traffic control are (not necessarily in this order):

1. Enable traffic control
2. Create policies
3. Populate the policies with classes and sub policies
4. Create rules to match traffic types and assign them to classes
5. Configure other aspects of traffic control, like interface virtual bandwidths, system class bandwidths etc.

Prioritising phone calls and video traffic

In this generic example configuration:

- The CIR value of 450mbit effectively puts a bandwidth limit on the top priority class.
- The voice and video classes each limit their own bandwidth, within this overall CIR of 450mbits.
- If the voice or video classes have packet to transmit, and have not currently exceeded their bandwidth limits (or the overall limit of the real-time traffic policy), then all other classes must wait.
- The 1mbit limit on the management class leaves plenty of bandwidth for the Internet access traffic.
- The Internet access policy is able to use up all the remaining bandwidth, and apportion it to file-transfer, web, and email traffic in the relative ratios 10:5:3.

```
traffic-control enable
 policy example-policy priority
  class first-class priority-level 10 sub-class-policy htb
   sub-class real-time-traffic cir 450mbit sub-sub-class-policy htb
    sub-sub-class voice cir 40mbit pir 50mbit preference 7
    sub-sub-class video cir 350mbit pir 400mbit preference 4
  class management priority-level 5 max 1mbit
  class internet-access priority-level sub-class-policy WRR
    sub-class file-transfer weight 10
    sub-class web-browsing weight 5
    sub-class email weight 3
rule 10 match app-voice from private.phone-vlan to Internet policy
Example-policy.First-class.real-time-traffic.voice
rule 15 match app-voice from Internet to private.phone-vlan policy
Example-policy.First-class.real-time-traffic.voice
rule 20 match app-video from private.head-end.video_server to
private.visual-display-vlan policy Example-policy.First-class.real-time-
traffic.videorule 110 match app to A policy system
rule 30 match ssh from private.IS-LAN to Private policy Example-
policy.management
```

```
rule 35 match ssh from Private to private.IS-LAN policy Example-
policy.management
rule 40 match SNMP from private.IS-LAN to Private policy Example-
policy.management
rule 45 match SNMP from Private to private.IS-LAN policy Example-
policy.management
rule 50 match HTTPS from private.IS-LAN to Private policy Example-
policy.management
rule 55 match HTTPS from Private to private.IS-LAN policy Example-
policy.management
rule 60 match TFTP from private.IS-LAN to Private policy Example-
policy.management
rule 65 match TFTP from Private to private.IS-LAN policy Example-
policy.management
rule 70 match FTP from private to Internet policy Example-policy.internet-
access.file-transfer
rule 75 match FTP from Internet to private policy Example-policy.
internet-access.file-transfer
rule 80 match HTTP from private to Internet policy Example-
policy.internet-access.web-browsing
rule 85 match HTTP from Internet to private policy Example-
policy.internet-access.web-browsing
rule 90 match HTTPS from private to Internet policy Example-
policy.internet-access.web-browsing
rule 95 match HTTPS from Internet to private policy Example-
policy.internet-access.web-browsing
rule 100 match email from private to Internet policy Example-
policy.internet-access.email
rule 105 match email from Internet to private policy Example-
policy.internet-access.email
```

Applying traffic control to bridging

Because traffic control must use the Zone, Network and Host entities to specify the egress interfaces on which to apply their queues, the queueing is applied to bridge instances, rather than interfaces within the bridge.

Example 1: Layer 3 routed traffic

This example applies a traffic control policy to traffic that is layer 3 routed from a VLAN (vlan3) into the bridge (br1).

```
!
zone Z
 network A
  ip subnet 192.168.1.0/24 interface br1
 network B
  ip subnet 192.168.2.0/24 interface vlan3

traffic-control
 policy P1 priority
  class C1 priority-level 15
 rule 10 match app from Z.B to Z.A policy P1.C1
 traffic-control enable
!
bridge 1
!
vlan database
 vlan 2-3 state enable
!
interface eth1
 bridge-group 1
!
interface vlan2
 bridge-group 1
!
interface vlan3
 ip address 192.168.2.1/24
!
interface br1
 ip address 192.168.1.1/24
```

Example 2: Layer 2 bridged traffic

This example applies a traffic control policy to traffic that is layer 2 bridged from VLAN (vlan1) to the eth1 interface.

From software version 5.4.7-0.1, traffic control is disabled by default for bridged traffic. As shown in the following example, you can enable it by using the following command in interface mode for the desired bridge:

```
awplus(config-if)#l3-filtering enable
```

We do not recommend shaping bridged traffic on firewalls that are running Unified Threat Management (UTM) features, because both Traffic Control and UTM features require significant CPU resources.

```
!
zone Z
 network A
  ip subnet 0.0.0.0/0 interface eth1
 network B
  ip subnet 0.0.0.0/0 interface vlan1

traffic-control
 policy P1 priority
  class C1 priority-level 15
 rule 10 match app from Z.B to Z.A policy P1.C1
 traffic-control enable
!
bridge 1
!
interface eth1
 bridge-group 1
!
interface vlan1
 bridge-group 1
!
interface br1
 l3-filtering enable
```

Applying traffic control to a tunnel

When applying traffic control to a tunnel, it is important to configure a virtual bandwidth to the tunnel interface, as tunnels do not have an inherent bandwidth.

```
zone private
 network lan
  ip subnet 10.37.249.32/27 interface eth2
 network tunnel
  ip subnet 10.37.36.64/27 interface tunnel1
!
traffic-control
 policy P priority
  class C1 priority-level 7
  class C2 priority-level 6 max 25mbit
 rule 10 match ftp from private.lan to private.tunnel policy P1.C1
 rule 20 match imap from private.lan to private.tunnel policy P.c2
 interface tunnel1 virtual-bandwidth 100mbit
 traffic-control enable
!
interface eth2
 ip address 10.37.249.34/27
!
interface tunnel1
  tunnel source 30.1.1.1
  tunnel destination 40.1.1.1
  tunnel mode gre
  ip address 172.168.1.1/24
```

Traffic control show commands

There are a number of show commands that you can use to verify and check information in relation to traffic control.

Note: The configured policy (as counted) includes the default policy that is automatically applied when traffic control is enabled.

show traffic-control

This command is used to see the status of traffic control, if it is enable or disabled and if there are policies or rules configured.

```
show traffic-control
Traffic control is enabled
Policy configured on 1013 interfaces (install/uninstall in progress: 287)
6 rules configured (2 valid rules)
Virtual-bandwidth configured on 3 interfaces
```

show traffic-control counters

This command is used to see how many packets or bytes of data have hit each installed traffic control policy, as well as the classes contained within the policy. Counters are maintained for classes at all levels of the hierarchy, including the root policy.

There are three counters for each class:

1. **Sent**—counts traffic that hit the class and has already been sent out the parent interface.
2. **Currently Queued**—counts traffic that is currently queued by the scheduling algorithm, and has not yet been sent.
3. **Dropped**—counts traffic that hit the class but was dropped for some reason, e.g. bandwidth is saturated.

Outgoing packets that are not matched by any rule go to the installed policy's default class. Packets egressing an interface that match rules using the system policy will go directly to the interface's system class.

```
show traffic-control counters
Traffic Control Counters

Interface eth1:
Class              Counter                    Bytes                Packets
-------------------------------------------------------------------------
P                  Sent                     12446671               45943
                   Currently Queued                0                   0
                   Dropped                                             8

P.C1               Sent                            0                   0
                   Currently Queued                0                   0
                   Dropped                                             0

P.C2               Sent                      8990019                5951
                   Currently Queued                0                   0
                   Dropped                                             8

P.default          Sent                      3456652               39992
                   Currently Queued                0                   0
                   Dropped                                             0

system             Sent                          516                   6
                   Currently Queued                0                   0
                   Dropped                                             0
```

```
Interface eth2:
Class              Counter                  Bytes                Packets
------------------------------------------------------------------------
P                  Sent                   118171308                81275
                   Currently Queued                0                   0
                   Dropped                                            25

P.C1               Sent                     26216057               17332
                   Currently Queued                0                   0
                   Dropped                                            11

P.C2               Sent                            0                   0
                   Currently Queued                0                   0
                   Dropped                                             0

P.default          Sent                     91955251               63943
                   Currently Queued                0                   0
                   Dropped                                            14

system             Sent                          516                   6
                   Currently Queued                0                   0
                   Dropped                                             0
```

show traffic-control interface

This command displays interface specific traffic control settings. Each interface has its own settings for virtual bandwidth, system class bandwidth, and packet overhead. The policy that is currently active on each interface is also shown. For descriptions of interface parameters, see the command **interface (traffic-control)**.

```
traffic-control
 policy P priority
  class C1 priority-level 15 max 50mbit
  class C2 priority-level 14 max 30mbit
 rule 10 match ftp from private.lan1 to private.lan2 policy P.C1
 rule 20 match ftp from private.lan2 to private.lan1 policy P.C2
 interface eth1 virtual-bandwidth 100mbit
 interface eth2 virtual-bandwidth 100mbit system-bandwidth 10 overhead 100
```

The output of the **show traffic control interface** command:

```
show traffic-control interface
Traffic Control Interface Information

eth1:
  Applied policy:       P
  Virtual bandwidth:    100000 kbit
  System bandwidth:     5%
  Packet overhead:      0 Bytes

eth2:
  Applied policy:       P
  Virtual bandwidth:    100000 kbit
  System bandwidth:     10%
  Packet overhead:      100 Bytes

vlan1:
  Applied policy:       Default policy (installation in progress)
  Virtual bandwidth:    Not set
  Packet overhead:      0 Bytes
```

Eth interfaces, as well as the default VLAN, will be present at startup. If they are not configured with a policy, they will be assigned a default policy (of which the underlying scheduling algorithm is FQ CoDel).

The example output shows that installation of the policy on vlan1 is still in progress. This would usually only be seen immediately after config changes or just after traffic control is enabled if there are a large number of interfaces. The policy is not fully operational on the interface until this message is no longer displayed.

show traffic-control policy

This command shows information about the configured traffic control policies as well as their class hierarchies and configured parameters. For each policy, a list is displayed of the interfaces that the policy is currently installed on.

Consider the case that an AR-Series Firewall has the following configuration:

```
zone A
 network ANET
  ip subnet 192.168.1.0/24 interface vlan2
!
zone B
 network BNET
  ip subnet 192.168.2.0/24 interface vlan3
!
application t0
 protocol tcp
 dport 5000
!
application t1
 protocol tcp
 dport 5001
!
traffic-control
 policy P priority
  class C1 priority-level 15 max 50mbit
  class C2 priority-level 6 sub-class-policy htb
   sub-class S1 cir 40mbit pir 60mbit set-dscp af11
 policy P2 wrr
  class C1 weight 50 sub-class-policy wrr
   sub-class S1 weight 30 sub-sub-class-policy wrr
    sub-sub-class SS1 weight 3
    sub-sub-class SS2 weight 4
   sub-class S2 weight 5
  class C2 weight 15
 rule 10 match t0 from B to A policy P.C1
 rule 20 match t1 from B to A policy P.C2.S1
 rule 30 match t0 from A to B policy P.C1
 interface vlan2 virtual-bandwidth 100mbit
 interface vlan3 virtual-bandwidth 100mbit
 traffic-control enable
```

The output of the **show traffic-control policy** command on this unit will be:

```
show traffic-control policy
Traffic Control Policies:

Policy P:
  Type:               priority
  Applied interfaces: vlan2 vlan3
  Classes:
    Class C1:
     Priority:             15
     Peak rate (PIR):      50000kbit
    Class C2:
     Priority:             6
     Sub-queue type:       htb
      Class S1:
       Committed rate (CIR): 40000kbit
       Peak rate (PIR):      60000kbit
       Set DSCP:             af11

Policy P2:
  Type:               wrr
  Applied interfaces: None
  Classes:
    Class C1:
     Weight:               50
     Sub-queue type:       wrr
      Class S1:
       Weight:               30
       Sub-queue type:       wrr
        Class SS1:
         Weight:               3
        Class SS2:
         Weight:               4
      Class S2:
       Weight:               5
    Class C2:
     Weight:               15
```

Note: In the above configuration, policy P2 is not used in any rules, therefore ‘Applied interfaces’ shows as ‘None’.

Show traffic-control rule

This command displays the configured traffic control rules in a tabular format. Consider the case that an AR-Series Firewall has the following configuration:

```
!
zone A
 network ANET
  ip subnet 192.168.1.0/24 interface vlan2
!
zone B
 network BNET
  ip subnet 192.168.2.0/24 interface vlan3
!
application t0
 protocol tcp
 dport 5000
!
application t1
 protocol tcp
 dport 5001
!
!
!
traffic-control
 policy P priority
  class C1 priority-level 15 max 50mbit
  class C2 priority-level 6 sub-class-policy htb
   sub-class S1 cir 40mbit pir 60mbit set-dscp af11
 policy P2 wrr
  class C1 weight 50 sub-class-policy wrr
   sub-class S1 weight 30 sub-sub-class-policy wrr
    sub-sub-class SS1 weight 3
    sub-sub-class SS2 weight 4
   sub-class S2 weight 5
  class C2 weight 15
 rule 10 match t0 from B to A policy P.C1
 rule 20 match t1 from B to A policy P.C2.S1
 rule 30 match t0 from A to B policy P.C1
 interface vlan2 virtual-bandwidth 100mbit
 interface vlan3 virtual-bandwidth 100mbit
 traffic-control enable
!
```

The output of the **show traffic-control rule** command on this unit will be:

```
show traffic-control rule
[* = Rule is not valid - see "show traffic-control rule config-check"]
  ID    App       From            To              Policy
----------------------------------------------------------------
  10    t0        B               A               P.C1
  20    t1        B               A               P.C2.S1
  30    t0        A               B               P.C1
```

If we reconfigure rule 20 using the following input:

```
awplus(config-tc)#no rule 20
awplus(config-tc)#rule 20 match t2 from B to A policy P.C2.S1
```

We get a warning message:

```
% Warning: Application t2 doesn't currently exist.
```

In the output of the **show traffic-control rule** command, rule 20 is now marked as invalid:

```
show traffic-control rule
[* = Rule is not valid - see "show traffic-control rule config-check"]
  ID    App       From            To              Policy
----------------------------------------------------------------
  10    t0        B               A               P.C1
* 20    t1        B               A               P.C2.S1
  30    t0        A               B               P.C1
```

To display more detailed information about an invalid rule, running the **show traffic-control rule config-check** command displays the reasons why the rule is invalid:

```
show traffic-control rule config-check
Rule 20:
  All rule parameters exist but rule is still invalid. Check that:
   - application has a protocol configured
   - entities have valid interfaces and subnets
```

Running the **show application detail** command shows that the application t2 has been registered (as a result of using it in a rule), but it does not have a protocol or any other attributes configured:

```
show application detail t2
Name                    Protocol       Detail
------------------------------------------------------------------------
t2                       -              -
```