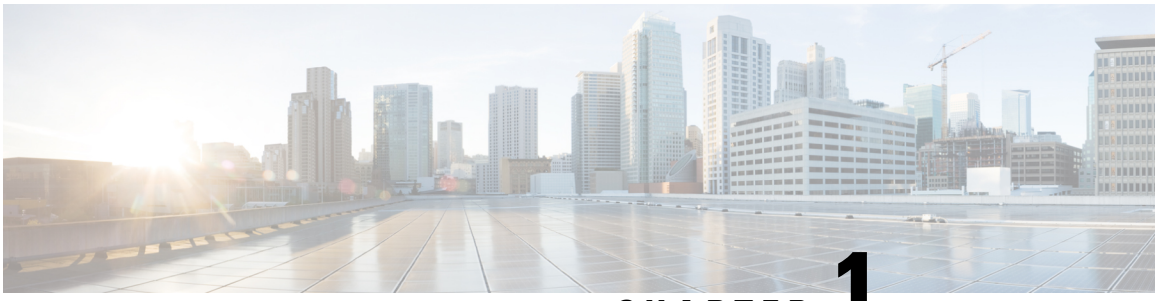
QoS Modular QoS Command-Line Interface Configuration Guide, Cisco IOS XE Gibraltar 16.12.x

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

CONTENTS



CHAPTER 1

Read Me First

Important Information about Cisco IOS XE 16

Effective Cisco IOS XE Release 3.7.0E for Catalyst Switching and Cisco IOS XE Release 3.17S (for Access and Edge Routing) the two releases evolve (merge) into a single version of converged release—the Cisco IOS XE 16—providing one release covering the extensive range of access and edge products in the Switching and Routing portfolio.

Feature Information

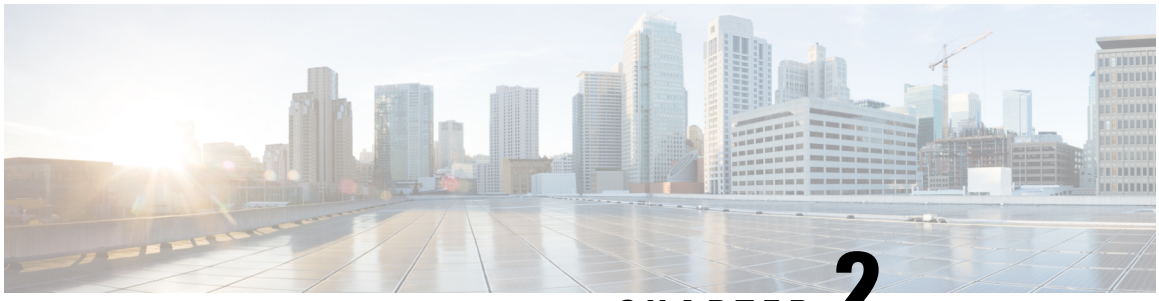
Use Cisco Feature Navigator to find information about feature support, platform support, and Cisco software image support. An account on Cisco.com is not required.

Related References

- Cisco IOS Command References, All Releases

Obtaining Documentation and Submitting a Service Request

- To receive timely, relevant information from Cisco, sign up at Cisco Profile Manager.
- To get the business impact you're looking for with the technologies that matter, visit Cisco Services.
- To submit a service request, visit Cisco Support.
- To discover and browse secure, validated enterprise-class apps, products, solutions and services, visit Cisco Marketplace.
- To obtain general networking, training, and certification titles, visit Cisco Press.
- To find warranty information for a specific product or product family, access Cisco Warranty Finder.



CHAPTER 2

Applying QoS Features Using the MQC

About

This chapter provide an overview of Modular QoS CLI (MQC), which is how all QoS features are configured on the Cisco ASR 1000 Series Aggregation Services Router. MQC is a standardized approach to enabling QoS on Cisco routing and switching platforms.

We intend this chapter as an overview of configuration tasks required for any QoS configuration. Individual features are covered in appropriate modules.

Cisco Modular QoS CLI

With MQC, you perform 4 simple steps to enable and verify QoS. Examples are shown for each step. (Refer to individual chapters for feature explanations.)

1. **Create class-maps** - Classify your traffic (applications) into classes that you will work on.

```
class-map voice
  match dscp ef
class-map video
  match dscp AF41 AF42
```

2. **Create policy-map** - Define the treatment each class should receive.

```
policy-map simple-example
  class voice
    priority
    police cir percent 10
  class video
    bandwidth remaining percent 30
```

3. **Attach the policy-map** - Bind the policy to a physical or logical interface, identifying the traffic on which your policy should operate. You must specify whether the policy will apply to traffic that will enter the router via that interface (ingress) or to traffic that will exit the router via that interface (egress).

```
interface gigabitethernet1/0/0
  service-policy out simple-example
```

4. **Verify operation of the QoS policy** - Issue the `show policy-map interface` command to verify operation of all QoS features configured with the MQC.

```
show policy-map interface gigabitethernet1/0/0
```

Create Class Maps

When you create a class-map you are defining a group of applications that should receive similar treatment. You will specify a name for the group and subsequently use that name when defining the treatment they should receive.

You will need to define one or more filters (classification rules), establishing that a particular packet (application) belongs to the group you specified. When you create a class-map, you can decide whether a packet must match just one filter (*match-any*) or all filters (*match-all*) to be considered part of that group.

Create a class-map as follows:

```
class-map [match-all|match-any] <traffic-class-name>
  match….    Filter1
  match…..   Filter2
```

The following example illustrates a class where a packet need only match a single filter. If either the packet has the DSCP value ef **or** Cisco NBAR recognizes that the packet carries the skype application, then we consider the packet *as belonging to the voice class*. We use the name voice in a policy-map to define treatment for any packet classified as belonging to this class:

```
class-map match-any voice
  match dscp ef
  match protocol skype
```

In the following example, we employ the match-all semantic: a packet must match all filters to belong to a class. We mandate that traffic must be recognized as MAPI (using Cisco NBAR) and also be to or from the address specified in the access list:

```
ip access-list extended mail-server-addr
  permit ip any host 10.10.10.1
  permit ip host 10.10.10.1 any
!
class-map match-all work-email
  match protocol mapi
  match access-group name mail-server-addr
```

The previous examples illustrate the flexibility of filter definitions on the ASR 1000 series platform. Filters can be based on marks in the packet header (precedence, DSCP, Exp or COS), access-lists, Cisco NBAR (match protocol *xxx*) or internal markings like qos-group. (Refer to the classification chapter for a more complete description of supported filters - when available.)

For convenience, you can also include other class-maps as filters in a class-map:

```
class-map broadcast-video
  match dscp cs5
class-map multimedia-streaming
  match dscp af31 af32 af33
class-map multimedia-conferencing
  match dscp af41 af42 af43
class-map realtime-interactive
  match dscp cs4
!
class-map match-any all-video
  match class broadcast-video
  match class multimedia-streaming
  match class multimedia conferencing
  match class realtime-interactive
!
class-map match-any interactive-video
  match class multimedia conferencing
  match class realtime-interactive
```

In this example we use *nested class-maps* in the definition of classes all-video and interactive-video.

By definition, a particular packet might match the classification criteria of multiple classes in a class-map. If so, the order in which classes are defined in a policy-map determines which class the packet belongs to; a packet belongs to the first class it matches.

Create Policy-Maps

A policy-map is how you specify what actions should apply to each class of traffic you create.

Let's re-examine the simple example above:

```
policy-map simple-example
  class voice
    priority
    police cir percent 10
  class video
    bandwidth remaining percent 30
```

The policy-map name is simple-example – this is the name we use when we subsequently attach the policy to one or more interfaces. The policy itself is quite readable – we have defined two classes of traffic: voice and video. Voice traffic should receive priority (low latency) scheduling but throughput of that class is limited to 10% of the interface bandwidth. For video traffic, we have a dedicated queue and a guarantee of 30% of what remains after voice is serviced.

The above policy-map has a 3rd implicit class; class-default is the last class in a policy, whether explicitly configured or not. It is a catch-all into which falls any traffic that does not match one of the user-defined classes. In egress policies class-default will have its own queue and an implicit bandwidth remaining ratio of 1. If bandwidth values are specified in percentage, class-default will receive any unassigned percent (see asterisks). Knowing this the above policy-map would actually look as follows:

```
class-map class-default
  match any
!
policy-map simple-example
  class voice
    priority
    police cir percent 10
```

```
  class video
    bandwidth remaining percent 30
  class class-default
    bandwidth remaining percent 70                ****
```

Note You never need to create a class-map for class-default. We visualize it here to provide a better understanding of how the policy works. If a packet does not match the voice class or the video class it will always match class-default.

Examples of actions in the policy above include the **priority**, **police**, and **bandwidth** commands. Actions function as control knobs to differentiate how one class of traffic will be treated vs. another.

One very important differentiation when looking at actions is queuing vs. non-queuing. What if we now add one more class to the simple-example policy-map:

```
class-map youtube
  match protocol youtube
  !
policy-map simple-example
  class voice
    priority
    police cir percent 10
  class youtube
    police cir percent 5
  class video
    bandwidth remaining percent 30
```

We have added a third user-defined class named youtube that is rate-limiting YouTube traffic such that it can never exceed 5% of link capacity. As this class has no queuing action configured, no queue is created (see below the list of actions that create a queue). Packets that match this class (those with protocol youtube) will traverse the policer and then be enqueued in the class-default queue.

Did you notice that we placed the youtube class before the video class in our policy definition? We want to ensure that youtube traffic is always part of this class rather than our video class. By defining this class earlier in the policy-map we will check for a match to this class before we check the video class criteria.

The specific actions that will create a queue are **priority**, **bandwidth**, **bandwidth remaining** and **shape**. Other actions like **fair-queue**, **queue-limit** and **random-detect** may only be used in a class already containing one of the actions that creates a queue. The actions **police** and **set** will not create a queue, although you can use the **police** command for queue admission control.

One key reason to differentiate between queuing actions and non-queuing actions is that a policy-map that will be applied to ingress traffic may not contain any queuing actions on the ASR 1000 Series Router. Let's summarize which actions are queuing and which are not:

Queuing and Non-Queuing Actions		**Action**
Queuing Actions		
	Scheduling	
		priority
		bandwidth

Queuing and Non-Queuing Actions		Action
		bandwidth remaining
		shape
		fair-queue
	Queue Management / Congestion Avoidance	
		queue-limit
		random-detect
Non-Queuing Actions		
	Rate-Limiting / Admission Control	
		police
	Marking	
		set

Hierarchical policy-maps can be created by embedding a policy-map within a class of another policy-map:

```
policy-map child
  class voice
    priority
    police cir percent 10
  class video
    bandwidth remaining percent 30
!
policy-map parent-vlan
  class class-default
    shape average 100m
    service-policy child
```

A common use is to create a *shape on parent* / *queue on child* policy that can be attached to a logical interface such as a VLAN or a tunnel.

From a classification perspective, a packet must adhere to the classification criteria of the child as well as the parent class to be considered a member of a particular child class. In this example the parent class is class-default and by definition any traffic will match this class.

When defining hierarchical polices we can re-use policy-maps for convenience.

In the following example, we use the policy-map named child in both parent-vlan100 and parent-vlan200. When instantiated (attached to an interface) the voice class in parent-vlan100 will be limited to 10 Mbps (10% of 100m parent shaper) while the voice class in parent-vlan200 will be limited to 5 Mbps (10% of 50m parent shaper):

```
policy-map child
  class voice
    priority
    police cir percent 10
```

```
  class video
    bandwidth remaining percent 30
!
policy-map parent-vlan100
  class class-default
    shape average 100m
    service-policy child
!
policy-map parent-vlan200
  class class-default
    shape average 50m
    service-policy child
!
int gigabitethernet1/0/0.100
  service-policy out parent-vlan100
int gigabitethernet1/0/0.200
  service-policy out parent-vlan200
```

This example shows that although the definition may be shared the instances of the policy on different interfaces are truly unique.

You can also create hierarchical policies with policy-maps used in user-defined classes.

The following example illustrates a 3-level hierarchical policy, the max currently supported on the ASR 1000 Series Router. For a packet to match a class at the application level it must now match 3 requirements: the voice or video classifier at the child, the vlan classifier in the vlan-sharing policy-map, and the class-default (anything) in the physical level policy-map:

```
class-map vlan100
  match vlan 100
class-map vlan200
  match vlan 200
!
policy-map child
  class voice
    priority
    police cir percent 10
  class video
    bandwidth remaining percent 30
!
policy-map vlan-sharing
  class vlan100
    shape average 100m
    service-policy child
  class vlan200
    shape average 50m
    service-policy child
!
policy-map physical-policy
  class class-default
    shape average 500m
    service-policy vlan-sharing
!
interface gigabitethernet1/0/0
  service-policy out physical-policy
```

When you create a policy-map, IOS will perform some error checking on the policy. For example, if I create a policy with an unconstrained priority queue and then guarantee bandwidth to another queue, IOS will recognize the disconnect; if the unconstrained priority queue can comsume the entire interface bandwidth then clearly you cannot guarantee any of that bandwidth to another queue:

```
policy-map create-error-example
  class unconstrained-priority
    priority
  class bandwidth-guarantee
    bandwidth percent 50
```

If IOS detects an error in the policy during creation, it will reject the configuration and display an error at that time.

Attach the Policy-Map

The third step in using the Cisco MQC is to *instantiate the policy-map* (ie., to attach the policy to an interface and thus initiate control of traffic). We use the **service-policy** command to attach the policy and also to specify whether it is acting on traffic ingressing that interface or egressing that interface:

```
interface gigabitethernet1/0/0
  service-policy out simple-example
```

We have already mentioned that queuing policies are only supported for egress traffic (**service-policy out** *policy-name*) but policies that contain only non-queuing actions may be attached for ingress (**service-policy in** *policy-name*) or egress traffic.

We term the interface to where we apply the **service-policy** command the *attach point*. This point could be a physical interface (such as an Ethernet interface or a T1 interface) or it could be a logical interface such as a VLAN sub-interface or a tunnel interface.

When a policy-map contains queuing actions but no hierarchical policies we refer to the policy as a *flat policy*. A flat policy may only be attached to a physical interface.

To attach a queuing policy to a logical interface, you must use a hierarchical shape on parent/queue on child style policy.

As you recall, error checking occurs when you create a policy. A second round of error checking occurs when you attach the policy to an interface. For example, I might create a policy with bandwidth guarantees that can't be realized on a particular type of interface. The policy-map may be valid when defined but when combined with information about the attachment interface, IOS can recognize the error as in the following example:

```
policy-map attach-error-example
  class bulk-data
    bandwidth 200000
```

This policy dictates that 200 Mbps should be reserved for bulk-data. If I attach this policy to a GigabitEthernet interface it should work fine. However, if I attach this policy to a POS OC3 interface it will be rejected at attach time. An OC3 interface has a nominal bandwidth of 155 Mbps. 200 Mbps could never be reserved for a particular class of traffic.

Verify Operation of the QoS Policy

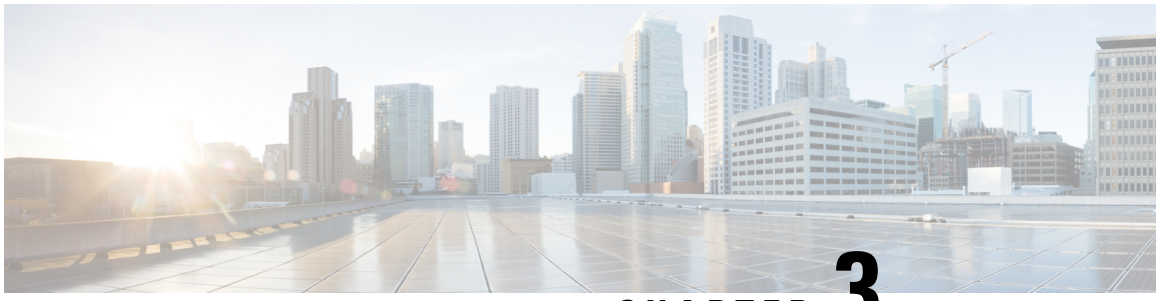
One command is always available to verify the operation of any QoS policy:

show policy-map interface *interface-name*

The output of this command displays a section for each class in a policy-map. It also shows statistics for packets and bytes classified as belonging to that class as well as for each action configured in the class.

Note The statistics available from this command are also available via SNMP in the CISCO-CLASS-BASED-QOS-MIB.

If the QoS policy is attached to a multipoint interface such as DMVPN, we use the **show policy-map multipoint tunnel** *tunnel-number* variant of the command. Similarly if the policy is attached to a broadband session, we would use the **show policy-map session uid** *session-number* variant of the command.



CHAPTER 3

3-Level User-Defined Queuing Policy Support

3-level user-defined queuing policy support feature allows 3 level policy with topmost layer user defined classes to support and enhance the flexibility of the traffic class in the hierarchy.

Finding Feature Information

Your software release may not support all the features documented in this module. For the latest caveats and feature information, see Bug Search Tool and the release notes for your platform and software release. To find information about the features documented in this module, and to see a list of the releases in which each feature is supported, see the feature information table.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to https://cfnng.cisco.com/. An account on Cisco.com is not required.

Restrictions for 3-Level User-Defined Queuing Policy Support

- User-defined class in top layer of a 3-level hierarchical queuing policy is not supported on port-channel main interface.

  User-defined class at the topmost layer is not supported on any logical target. Logical targets include service-group, tunnel, session, dealer interface, etc.

Information About 3-Level User-Defined Queuing Policy Support

Three-Parameter Scheduler in Hierarchical QoS

Classic IOS uses max value (shape) and min value (bandwidth) to define each scheduler node behavior when traffic congestion happens, or 2 parameter scheduler.

ASR 1000 utilize a different 3-parameter scheduler: max value (shape), min value (bandwidth) and excess value (bandwidth remaining) which is to adjust sharing of excess bandwidth. In a 2-parameter scheduler, the excess bandwidth are shared by the classes proportionally (same as the bandwidth ratio for each class); But in a 3-parameter scheduler, the excess bandwidth are shared equally by default after satisfying minimum bandwidth requirements, but it can be tuned when using 'bandwidth remaining' command. ISR 4000 platforms share the same design.

In Classic IOS, it is permitted to configure bandwidth at the leaf and intermediate nodes of a hierarchy. In IOS XE, bandwidth (bandwidth rate , or bandwidth percent) is only allowed at the leaf node of a hierarchy. In other words, bandwidth (bandwidth rate , or bandwidth percent) class cannot be attached with a child policy-map containing queuing features. This is a restriction in software and may be lifted in the future.

For current deployments where a Classic IOS QoS policy-map is being moved to a IOS XE platform, the best option is to convert the intermediate node bandwidth commands to bandwidth remaining commands. bandwidth remaining percent or bandwidth remaining ratio commands could be used to achieve very similar behavior.

Guidelines for Hierarchical Policies

In general, three levels of hierarchy are supported on ASR 1000. Hierarchical policy can be applied on most of the physical and logical targets that supports QoS.

If you mix queuing and non-queuing policies together in a hierarchy, the non-queuing policy-maps must be at the leaf level of the policy-map (for example, child policy beneath grandparent and parent queuing policies).

If the policy-map is applied to a virtual interface (such as a tunnel or session), there may be additional restrictions limiting the hierarchy to two levels of queuing, depending on the configuration.

- Queuing features: shape, bandwidth, bandwidth remaining, random-detect, fair-queue, queue limit, and priority.
- Non-queuing features: police, mark, and account.

Hierarchy Feature Combination	Ingress Policy Support	Egress Policy Support
One-level Non-queuing Policy	Yes	Yes
Two-level Non-queuing Policy (including color-aware police)	Yes	Yes
Three-level Non-queuing Policy (including hierarchical color-aware police)	Yes	Yes
One-level Queuing Policy	-	Yes

Hierarchy Feature Combination	Ingress Policy Support	Egress Policy Support
Two-level Queuing Policy	-	Yes
Three-level Queuing Policy	-	Yes
Two-level Mixed Policy, Queuing feature at parent level	-	Yes
Three-level Mixed Policy, Queuing feature at grandparent level, or at grandparent + parent level	-	Yes

User-defined Traffic Class in Top-level Policy of HQoS

Any traffic class configured explicitly by 'class-map' is called 'user-defined class'. Class-default classes need not be configured, and can be used in any policy to match all the traffic that does not belong to user-defined classes.

In a three-level queuing policy-map, only class-default class can be configured in the highest level before Release Polaris 16.3. From Polaris 16.3, user-defined class in top layer of a 3-level hierarchical policy is supported.

How to Configure 3-Level User-Defined Queuing Policy Support

Configuring 3-level Hierarchical QoS Policy

```
enable
configure terminal
class-map vlan10
 match vlan10
class-map vlan20
 match vlan 20
class-map ef
 match dscp ef
policy-map child
 class ef
  priority
  police 1000000
 class class-default
 police 3000000
policy-map parent
 class vlan10
  shape average 4000000
  service-policy child
 class vlan20
  shape average 8000000
  service-policy child
policy-map grand-parent
 class class-default
  shape average 10000000
  service-policy parent
end
```

Configuring User-Defined Traffic Class in Top Level Policy

```
ip access-list extended PEER
permit ip host 200.0.0.2 any

class-map match-all ef
 match dscp ef
class-map match-all vlan100
 match vlan 100
class-map match-all vlan101
 match vlan 101
class-map match-all PEER
 match access-group name PEER

policy-map child
 class ef
  bandwidth remaining percent 15
 class class-default
  fair-queue
  queue-limit 512 packets
  bandwidth remaining percent 85

policy-map parent
 class PEER
  shape average 8000000
  bandwidth remaining percent 10
  service-policy child
class class-default
 shape average 8000000

policy-map grandparent
 class vlan100
  shape average 8000000
  bandwidth remaining ratio 1000
  service-policy parent
 class vlan101
  shape average 8000000
  bandwidth remaining ratio 1000
  service-policy parent
 class class-default
  bandwidth remaining ratio 1
  shape average 10000000
end
```

Additional References for 3-Level User-Defined Queuing Policy Support

Related Documents

Related Topic	Document Title
Cisco IOS commands	Cisco IOS Master Commands List, All Releases

Technical Assistance

Description	Link
The Cisco Support website provides extensive online resources, including documentation and tools for troubleshooting and resolving technical issues with Cisco products and technologies. To receive security and technical information about your products, you can subscribe to various services, such as the Product Alert Tool (accessed from Field Notices), the Cisco Technical Services Newsletter, and Really Simple Syndication (RSS) Feeds. Access to most tools on the Cisco Support website requires a Cisco.com user ID and password.	http://www.cisco.com/cisco/web/support/index.html

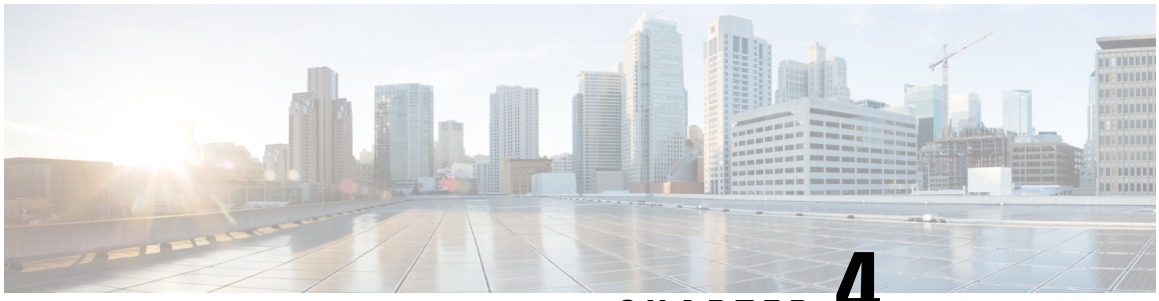
Feature Information for 3-Level User-Defined Queuing Policy Support

The following table provides release information about the feature or features described in this module. This table lists only the software release that introduced support for a given feature in a given software release train. Unless noted otherwise, subsequent releases of that software release train also support that feature.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to www.cisco.com/go/cfn. An account on Cisco.com is not required.

Table 1: Feature Information for 3-Level User-Defined Queuing Policy Support

Feature Name	Releases	Feature Information
3-Level User-Defined Queuing Policy Support	Cisco IOS XE Denali 16.3.1.	This feature is introduced on Cisco ASR 1000, ISR4000, CSR1000v platforms. User-defined class can be configured in top layer of a 3-level hierarchical policy.



CHAPTER 4

Configuring IP to ATM Class of Service

This module describes the tasks for configuring the IP to ATM Class of Service (CoS), a feature suite that maps QoS characteristics between IP and ATM.

Use Cisco Feature Navigator to find information about platform support and software image support. Cisco Feature Navigator enables you to determine which Cisco IOS and Catalyst OS software images support a specific software release, feature set, or platform. To access Cisco Feature Navigator, go to http://www.cisco.com/go/cfn . An account on Cisco.com is not required.

Finding Feature Information

Your software release may not support all the features documented in this module. For the latest caveats and feature information, see Bug Search Tool and the release notes for your platform and software release. To find information about the features documented in this module, and to see a list of the releases in which each feature is supported, see the feature information table.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to https://cfnng.cisco.com/. An account on Cisco.com is not required.

IP to ATM CoS on a Single ATM VC Configuration Task List

To configure IP to ATM CoS for a single ATM virtual circuit (VC), perform the tasks described in the following sections. The tasks in the first two sections are required; the tasks in the remaining sections are optional.

The IP to ATM CoS feature requires ATM permanent virtual circuit (PVC) management.

Defining the WRED Parameter Group

Command	Purpose
`Router(config)# random-detect-group group-name`	Defines the WRED or VIP-distributed WRED (DWRED) parameter group.

Configuring the WRED Parameter Group

SUMMARY STEPS

1. Device(config)# **random-detect-group** *group-name*
2. Device(config)# **exponential-weighting-constant** *exponent*
3. Device(config)# **precedence** *precedence min-threshold max-threshold mark-probability-denominator*

DETAILED STEPS

	Command or Action	Purpose
Step 1	Device(config)# **random-detect-group** *group-name*	Specifies the WRED or DWRED parameter group.
Step 2	Device(config)# **exponential-weighting-constant** *exponent*	Configures the exponential weight factor for the average queue size calculation for the specified WRED or DWRED parameter group. or
Step 3	Device(config)# **precedence** *precedence min-threshold max-threshold mark-probability-denominator*	Configures the specified WRED or DWRED parameter group for a particular IP precedence.

Displaying the WRED Parameters

Command	Purpose
`Router# show queueing random-detect [interface atm_subinterface [vc [[vpi/] vci]]]`	Displays the parameters of every VC with WRED or DWRED enabled on the specified ATM subinterface.

Displaying the Queueing Statistics

Command	Purpose
`Router# show queueing interface interface-number [vc [[vpi/] vci]]`	Displays the queueing statistics of a specific VC on an interface.

IP to ATM CoS on an ATM Bundle Configuration Task List

To configure IP to ATM CoS on an ATM bundle, perform the tasks in the following sections.

The IP to ATM CoS feature requires ATM PVC management.

Creating a VC Bundle

Command	Purpose
`Router(config-subif)# bundle bundle-name`	Creates the specified bundle and enters bundle configuration mode.

Applying Bundle-Level Parameters

Configuring Bundle-Level Parameters

Command	Purpose
`Device(config-atm-bundle)# protocol protocol {protocol-address | inarp} [[no] broadcast]`	Configures a static map or enables Inverse Address Resolution Protocol (Inverse ARP) or Inverse ARP broadcasts for the bundle. **Note** Bundle-level parameters can be applied either by assigning VC classes or by directly applying them to the bundle. Parameters applied through a VC class assigned to the bundle are superseded by those applied at the bundle level. Bundle-level parameters are superseded by parameters applied to an individual VC.
`Device(config-atm-bundle)# encapsulation aal-encap`	Configures the ATM adaptation layer (AAL) and encapsulation type for the bundle.
`Device(config-atm-bundle)# inarp minutes`	Configures the Inverse ARP time period for all VC bundle members.
`Device(config-atm-bundle)# broadcast`	Enables broadcast forwarding for all VC bundle members.
`Device(config-atm-bundle)# oam retry up-count down-count retry frequency`	Configures the VC bundle parameters related to operation, administration, and maintenance (OAM) management.
`Device(config-atm-bundle)# oam-bundle [manage] [frequency]`	Enables end-to-end F5 OAM loopback cell generation and OAM management for all VCs in the bundle.

Configuring VC Class Parameters to Apply to a Bundle

Command	Purpose
`Router(config-vc-class)# oam-bundle [manage] [frequency]`	Enables end-to-end F5 OAM loopback cell generation and OAM management for all VCs in the bundle. **Note** Use of a VC class allows you to configure a bundle applying multiple attributes to it at once because you apply the class itself to the bundle. Use of a class allows you to generalize a parameter across all VCs, after which (for some parameters) you can modify that parameter for individual VCs. (See the section "Applying Parameters to Individual VCs" for more information.)

Attaching a Class to a Bundle

Command	Purpose
`(config-atm-bundle)# class-bundle vc-class-name`	Configures a bundle with the bundle-level commands contained in the specified VC class. **Note** Parameters set through bundle-level commands contained in the VC class are applied to the bundle and all of its VC members. Bundle-level parameters applied through commands configured directly on the bundle supersede those applied through a VC class. Some bundle-level parameters applied through a VC class or directly to the bundle can be superseded by commands that you directly apply to individual VCs in bundle-vc configuration mode.

Committing a VC to a Bundle

Command	Purpose
`Device(config-atm-bundle)# pvc-bundle pvc-name [vpi/] [vci]`	Adds the specified VC to the bundle and enters bundle-vc configuration mode in order to configure the specified VC bundle member.

Applying Parameters to Individual VCs

Configuring a VC Bundle Member Directly

Command	Purpose
`Device(config-if-atm-member)# ubr output-pcr [input-pcr]`	Configures the VC for unspecified bit rate (UBR) QoS and specifies the output peak cell rate (PCR) for it.

Command	Purpose
`Device(config-if-atm-member)# ubr+ output-pcr output-mcr [input-pcr] [input-mcr]`	Configures the VC for UBR QoS and specifies the output PCR and output minimum guaranteed cell rate for it.
`Device(config-if-atm-member)# vbr-nrt output-pcr output-scr output-mbs [input-pcr] [input-scr] [input-mbs]`	Configures the VC for variable bit rate nonreal-time (VBR-nrt) QoS and specifies the output PCR, output sustainable cell rate, and output maximum burst cell size for it.
`Device(config-if-atm-member)# precedence [other | range]`	Configures the precedence levels for the VC.
`Device(config-if-atm-member)# bump {implicit | explicit precedence-level | traffic}`	Configures the bumping rules for the VC.
`Device(config-if-atm-member)# protect {group | vc}`	Configures the VC to belong to the protected group of the bundle or to be an individually protected VC bundle member.

Configuring VC Class Parameters to Apply to a VC Bundle Member

Command	Purpose
`Device(config-vc-class)# bump {implicit | explicit precedence-level | traffic}`	Specifies the bumping rules for the VC member to which the class is applied. These rules determine to which VC in the bundle traffic is directed when the carrier VC bundle member goes down. **Note** You can also add the following commands to a VC class to be used to configure a VC bundle member: **ubr**, **ubr+**, and **vbr-nrt**. When a VC is a member of a VC bundle, the following commands cannot be used in vc-class mode to configure the VC: **encapsulation**, **protocol**, **inarp**, and **broadcast**. These commands are useful only at the bundle level, not the bundle member level. Configuration for an individual VC overrides the collective configuration applied to all VC bundle members through application of a VC class to the bundle.
`Device(config-vc-class)# precedence precedence min-threshold max-threshold mark-probability-denominator`	Defines precedence levels for the VC member to which the class is applied.
`Device(config-vc-class)# protect {group | vc}`	Configures the VC as a member of the protected group of the bundle or as an individually protected VC.

Applying a VC Class to a Discrete VC Bundle Member

Command	Purpose
`Device(config-if-atm-member)# class-vc vc-class-name`	Assigns a VC class to a VC bundle member.

Configuring a VC Not to Accept Bumped Traffic

Command	Purpose
`Device(config-if-atm-member)# no bump traffic`	Configures the VC not to accept any bumped traffic that would otherwise be redirected to it.

Monitoring and Maintaining VC Bundles and Their VC Members

Command	Purpose
`Device# show atm bundle bundle-name`	Displays the bundle attributes assigned to each bundle VC member and the current working status of the VC members.
`Device# show atm bundle bundle-name statistics [detail]`	Displays statistics or detailed statistics on the specified bundle.
`Device# show atm map`	Displays a list of all configured ATM static maps to remote hosts on an ATM network and on ATM bundle maps.
`Device# debug atm bundle errors`	Displays information on bundle errors.
`Device# debug atm bundle events`	Displays a record of bundle events.

Per-VC WFQ and CBWFQ Configuration Task List

To configure IP to ATM CoS for per-VC WFQ and CBWFQ, perform the tasks described in the following sections.

The IP to ATM CoS feature requires ATM PVC management.

Configuring Class-Based Weighted Fair Queueing

Before configuring CBWFQ for a VC, you must perform the following tasks using standard CBWFQ commands:

- Create one or more classes to be used to classify traffic sent across the VC

- Define a policy-map containing the classes to be used as the service policy

Note You can configure class policies for as many classes as are defined on the router, up to the maximum of 64. However, the total amount of bandwidth allocated for all classes included in a policy-map to be attached to a VC must not exceed 75 percent of the available bandwidth of the VC. The remaining 25 percent of available bandwidth is used for encapsulation, such as the ATM cell overhead (also referred to as ATM cell tax), routing and best-effort traffic, and other functions that assume overhead. For more information on bandwidth allocation, see the "Congestion Management Overview" module.

Because CBWFQ gives you minimum bandwidth guarantee, you can only apply CBWFQ to VCs having these classes of service: available bit rate (ABR) and variable bit rate (VBR). You cannot apply per-VC WFQ and CBWFQ to UBR and unspecified bit rate plus (UBR+) VCs because both of these service classes are best-effort classes that do not guarantee minimum bandwidth. When CBWFQ is enabled for a VC, all classes configured as part of the service policy are installed in the fair queueing system.

In addition to configuring CBWFQ at the VC level, the IP to ATM CoS feature allows you to configure flow-based WFQ at the VC level. Because flow-based WFQ gives you best-effort class of service--that is, it does not guarantee minimum bandwidth--you can configure per-VC WFQ for all types of CoS VCs: ABR, VBR, UBR, and UBR+.

Per-VC WFQ uses the class-default class. Therefore, to configure per-VC WFQ, you must first create a policy-map and configure the class-default class. (You need not create the class-default class, which is predefined, but you must configure it.) For per-VC WFQ, the class-default class must be configured with the **fair-queue** policy-map class configuration command.

In addition to configuring the **fair-queue** policy-map class configuration command, you can configure the default class with either the **queue-limit** command or the **random-detect**command, but not both. Moreover, if you want the default class to use flow-based WFQ, you cannot configure the default class with the **bandwidth** policy-map class configuration command--to do so would disqualify the default class as flow-based WFQ, and therefore limit application of the service policy containing the class to ABR and VBR VCs.

Attaching a Service Policy and Enabling CBWFQ for a VC

Attaching a Policy-Map to a Standalone VC and Enabling CBWFQ

Command	Purpose
`Router(config-if-atm-vc)#` **`service-policy output`** *`policy-map`*	Enables CBWFQ and attaches the specified service policy-map to the VC being created or modified.

Attaching a Policy-Map to an Individual VC and Enabling CBWFQ

Command	Purpose
`Router(config-if-atm-member)#` **`service-policy output`** *`policy-map`*	Enables CBWFQ and attaches the specified service policy-map to the VC being created or modified.

Note The **service-policy output** and **random-detect-group** commands are mutually exclusive; you cannot apply a WRED group to a VC for which you have enabled CBWFQ through application of a service policy. Moreover, before you can configure one command, you must disable the other if it is configured.

Configuring a VC to Use Flow-Based WFQ

SUMMARY STEPS

1. Device(config)# **policy-map** *policy-map*
2. Device(config-pmap)# **class class-default** *default-class-name*
3. Device(config-pmap-c)# **fair-queue** *number-of-dynamic-queues*
4. Do one of the following:
   - Device(config-pmap-c)# **queue-limit** *number-of-packets*
   - Device(config-pmap-c)# **random-detect**

DETAILED STEPS

	Command or Action	Purpose
Step 1	Device(config)# **policy-map** *policy-map*	Specifies the name of the policy-map to be created or modified.
Step 2	Device(config-pmap)# **class class-default** *default-class-name*	Specifies the default class so that you can configure or modify its policy. **Note** You can include other classes in the same policy-map as the one that contains the flow-based WFQ class. Packets not otherwise matched are selected by the default class-default class match criteria.
Step 3	Device(config-pmap-c)# **fair-queue** *number-of-dynamic-queues*	Specifies the number of dynamic queues to be reserved for use by flow-based WFQ running on the default class. **Note** By default--that is, even if you do not configure the class-default class with the **fair-queue** policy-map class configuration command and you do not configure it with the **bandwidth** policy-map class configuration command--the default class is defined as flow-based WFQ.
Step 4	Do one of the following: • Device(config-pmap-c)# **queue-limit** *number-of-packets* • Device(config-pmap-c)# **random-detect**	Specifies the maximum number of packets that can be queued for the class. Enables WRED. The class policy will drop packets using WRED instead of tail drop.

Attaching a Policy-Map to a Standalone VC and Enabling WFQ

Command	Purpose
`Device(config-if-atm-vc)# service-policy output policy-map`	Enables WFQ for the VC by attaching the specified policy-map containing the class-default class to the VC being created or modified.

Attaching a Policy-Map to an Individual VC and Enabling WFQ

Command	Purpose
`Device(config-if-atm-member)# service-policy output policy-map`	Enables WFQ for the VC bundle member by attaching the specified policy-map containing the class-default class to the VC bundle member.

Monitoring per-VC WFQ and CBWFQ

Command	Purpose
`Device# show policy-map interface interface-number [vc [vpi/] vci]]`	Displays the contents of packets inside a queue for a particular interface or VC.

Enabling Logging of Error Messages to the Console

Command	Purpose
`Router(config)# logging console level`	Limits messages logged to the console based on severity.

IP to ATM CoS Configuration Examples

Example Single ATM VC with WRED Group and IP Precedence

The following example creates a PVC on an ATM interface and applies the WRED parameter group called sanjose to that PVC. Next, the IP Precedence values are configured for the WRED parameter group sanjose.

```
interface ATM1/1/0.46 multipoint
 ip address 200.126.186.2 255.255.255.0
 no ip mroute-cache
 shutdown
pvc 46
 encapsulation aal5nlpid
 random-detect attach sanjose
!
random-detect-group sanjose
```

```
 precedence 0  200   1000 10
 precedence 1  300   1000 10
 precedence 2  400   1000 10
 precedence 3  500   1000 10
 precedence 4  600   1000 10
 precedence 5  700   1000 10
 precedence 6  800   1000 10
 precedence 7  900   1000 10
```

Example VC Bundle Configuration Using a VC Class

This example configures VC bundle management on a router that uses Intermediate System-to-Intermediate System (IS-IS) as its IP routing protocol.

Bundle-Class Class

At the outset, this configuration defines a VC class called bundle-class that includes commands that set VC parameters. When the class bundle-class is applied at the bundle level, these parameters are applied to all VCs that belong to the bundle. Note that any commands applied directly to an individual VC of a bundle in bundle-vc mode take precedence over commands applied globally at the bundle level. Taking into account hierarchy precedence rules, VCs belonging to any bundle to which the class bundle-class is applied will be characterized by these parameters: aal5snap encapsulation, broadcast on, use of Inverse Address Resolution Protocol (ARP) to resolve IP addresses, and operation, administration, and maintenance (OAM) enabled.

```
router isis
 net 49.0000.0000.0000.1111.00
vc-class atm bundle-class
 encapsulation aal5snap
 broadcast
 protocol ip inarp
 oam-bundle manage 3
 oam retry 4 3 10
```

The following sections of the configuration define VC classes that contain commands specifying parameters that can be applied to individual VCs in a bundle by assigning the class to that VC.

Control-Class Class

When the class called control-class is applied to a VC, the VC carries traffic whose IP Precedence level is 7. When the VC to which this class is assigned goes down, it takes the bundle down with it because this class makes the VC a protected one. The QoS type of a VC using this class is vbr-nrt.

```
vc-class atm control-class
 precedence 7
 protect vc
 vbr-nrt 10000 5000 32
```

Premium-Class Class

When the class called premium-class is applied to a VC, the VC carries traffic whose IP Precedence levels are 6 and 5. The VC does not allow other traffic to be bumped onto it. When the VC to which this class is applied goes down, its bumped traffic will be redirected to a VC whose IP Precedence level is 7. This class makes a VC a member of the protected group of the bundle. When all members of a protected group go down, the bundle goes down. The QoS type of a VC using this class is vbr-nrt.

```
vc-class atm premium-class
 precedence 6-5
 no bump traffic
 protect group
 bump explicitly 7
 vbr-nrt 20000 10000 32
```

Priority-Class Class

When the class called priority-class is applied to a VC, the VC is configured to carry traffic with IP Precedence in the 4-2 range. The VC uses the implicit bumping rule, it allows traffic to be bumped, and it belongs to the protected group of the bundle. The QoS type of a VC using this class is ubr+.

```
vc-class atm priority-class
 precedence 4-2
 protect group
 ubr+ 10000 3000
```

Basic-Class Class

When the class called basic-class is applied to a VC, the VC is configured through the **precedence other** command to carry traffic with IP Precedence levels not specified in the profile. The VC using this class belongs to the protected group of the bundle. The QoS type of a VC using this class is ubr.

```
vc-class atm basic-class
 precedence other
 protect group
 ubr 10000
```

The following sets of commands configure three bundles that the router subinterface uses to connect to three of its neighbors. These bundles are called new-york, san-francisco, and los-angeles. Bundle new-york has four VC members, bundle san-francisco has four VC members, and bundle los-angeles has three VC members.

new-york Bundle

The first part of this example specifies the IP address of the subinterface, the router protocol--the router uses IS-IS as an IP routing protocol--and it creates the first bundle called new-york and enters bundle configuration mode:

```
interface atm 1/0.1 multipoint
 ip address 10.0.0.1 255.255.255.0
 ip router isis
 bundle new-york
```

From within bundle configuration mode, the next portion of the configuration uses two protocol commands to enable IP and Open Systems Interconnect (OSI) traffic flows in the bundle. The OSI routing packets will use the highest precedence VC in the bundle. The OSI data packets, if any, will use the lowest precedence VC in the bundle. If configured, other protocols, such as IPX or AppleTalk, will always use the lowest precedence VC in the bundle.

As the indentation levels of the preceding and following commands suggest, subordinate to bundle new-york is a command that configures its protocol and a command that applies the class called bundle-class to it.

```
  protocol ip 1.1.1.2 broadcast
```

```
 protocol clns 49.0000.0000.2222.00 broadcast
 class-bundle bundle-class
```

The class called bundle-class, which is applied to the bundle new-york, includes a **protocol ip inarp** command. According to inheritance rules, **protocol ip**, configured at the bundle level, takes precedence over **protocol ip inarp** specified in the class bundle-class.

The next set of commands beginning with **pvc-bundle ny-control 207**, which are further subordinate, add four VCs (called ny-control, ny-premium, ny-priority, and ny-basic) to the bundle new-york. A particular class--that is, one of the classes predefined in this configuration example--is applied to each VC to configure it with parameters specified by commands included in the class.

As is the case for this configuration, to configure individual VCs belonging to a bundle, the router must be in bundle mode for the mother bundle. For each VC belonging to the bundle, the subordinate mode is pvc-mode for the specific VC.

The following commands configure the individual VCs for the bundle new-york:

```
pvc-bundle ny-control 207
 class-vc control-class
pvc-bundle ny-premium 206
 class-vc premium-class
pvc-bundle ny-priority 204
 class-vc priority-class
pvc-bundle ny-basic 201
 class-vc basic-class
```

san-francisco Bundle

The following set of commands create and configure a bundle called san-francisco. At the bundle configuration level, the configuration commands included in the class bundle-class are ascribed to the bundle san-francisco and to the individual VCs that belong to the bundle. Then, the **pvc-bundle** command is executed for each individual VC to add it to the bundle. After a VC is added and bundle-vc configuration mode is entered, a particular, preconfigured class is assigned to the VC. The configuration commands comprising that class are used to configure the VC. Rules of hierarchy apply at this point. Command parameters contained in the applied class are superseded by the same parameters applied at the bundle configuration level, which are superseded by the same parameters applied directly to a VC.

```
bundle san-francisco
 protocol clns 49.0000.0000.0000.333.00 broadcast
 inarp 1
 class-bundle bundle-class
 pvc-bundle sf-control 307
  class-vc control-class
 pvc-bundle sf-premium 306
  class-vc premium-class
 pvc-bundle sf-priority 304
  class-vc priority-class
 pvc-bundle sf-basic 301
  class-vc basic-class
```

los-angeles Bundle

The following set of commands create and configure a bundle called los-angeles. At the bundle configuration level, the configuration commands included in the class bundle-class are ascribed to the bundle los-angeles and to the individual VCs that belong to the bundle. Then, the **pvc-bundle** command is executed for each individual VC to add it to the bundle. After a VC is added and bundle-vc configuration mode is entered,

precedence is set for the VC and the VC is either configured as a member of a protected group (protect group) or as an individually protected VC. A particular class is then assigned to each VC to further characterize it. Rules of hierarchy apply. Parameters of commands applied directly and discretely to a VC take precedence over the same parameters applied within a class to the VC at the bundle-vc configuration level, which take precedence over the same parameters applied to the entire bundle at the bundle configuration level.

```
bundle los-angeles
 protocol ip 1.1.1.4 broadcast
 protocol clns 49.0000.0000.4444.00 broadcast
 inarp 1
 class-bundle bundle-class
 pvc-bundle la-high 407
  precedence 7-5
  protect vc
  class-vc premium-class
 pvc-bundle la-mid 404
  precedence 4-2
  protect group
  class-vc priority-class
 pvc-bundle la-low 401
  precedence other
  protect group
  class-vc basic-class
```

Example Per-VC WFQ and CBWFQ on a Standalone VC

The following example creates two class maps and defines their match criteria. For the first map class, called class1, the numbered access control list (ACL) 101 is used as the match criterion. For the second map class called class2, the numbered ACL 102 is used as the match criterion.

Next, the example includes these classes in a policy-map called policy1. For class1, the policy includes a minimum bandwidth allocation request of 500 kbps and maximum packet count limit of 30 for the queue reserved for the class. For class2, the policy specifies only the minimum bandwidth allocation request of 1000 kbps, so the default queue limit of 64 packets is assumed. Note that the sum of the bandwidth requests for the two classes comprising policy1 is 75 percent of the total amount of bandwidth (2000 kbps) for the PVC called cisco to which the policy-map is attached.

The example attaches the policy-map called policy1 to a PVC. Once the policy-map policy1 is attached to the PVC, its classes constitute the CBWFQ service policy for that PVC. Packets sent on this PVC will be checked for matching criteria against ACLs 101 and 102 and classified accordingly.

Because the **class-default** command is not explicitly configured for this policy-map, all traffic that does not meet the match criteria of the two classes comprising the service policy is handled by the predefined class-default class, which provides best-effort flow-based WFQ.

```
class-map class1
 match access-group 101
class-map class2
 match access-group 102
policy-map policy1
 class class1
  bandwidth 500
  queue-limit 30
 class class2
  bandwidth 1000
interface ATM1/1/0.46 multipoint
 ip address 200.126.186.2 255.255.255.0
 pvc 46
```

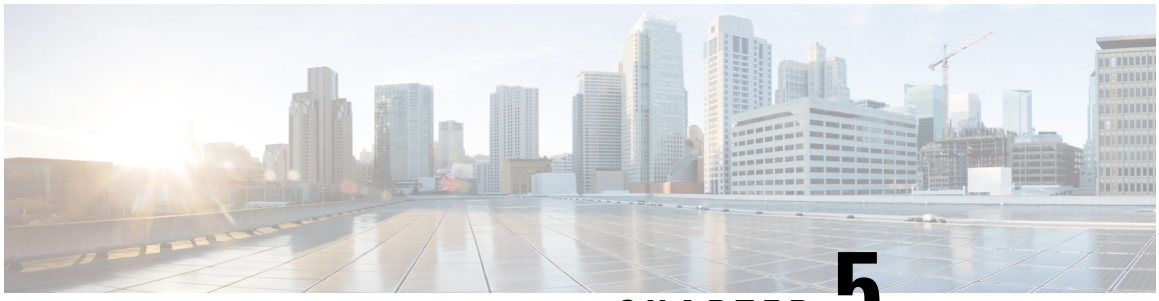
```
 vbr-nrt 2000 2000
 encap aal5snap
 service policy output policy1
```

Example Per-VC WFQ and CBWFQ on Bundle-Member VCs

The following example shows a PVC bundle called san-francisco with members for which per-VC WFQ and CBWFQ are enabled and service policies configured. The example assumes that the classes included in the following policy-maps have been defined and that the policy-maps have been created: policy1, policy2, and policy4. For each PVC, the IP to ATM CoS **pvc-bundle** command is used to specify the PVC to which the specified policy-map is to be attached.

Note that PVC 0/34 and 0/31 have the same policy-map attached to them, policy2. Although you can assign the same policy-map to multiple VCs, each VC can have only one policy-map attached at an output PVC.

```
bundle san-francisco
 protocol ip 1.0.2.20 broadcast
 encapsulation aal5snap
 pvc-bundle 0/35
  service policy output policy1
  vbr-nrt 5000 3000 500
  precedence 4-7
 pvc-bundle 0/34
  service policy output policy2
  vbr-nrt 5000 3000 500
  precedence 2-3
 pvc-bundle 0/33
  vbr-nrt 4000 3000 500
  precedence 2-3
  service policy output policy4
 pvc-bundle 0/31
  service policy output policy2
```



CHAPTER 5

QoS Scheduling

This chapter outlines the process of selecting the next packet to exit an interface and when it should happen (henceforth termed Scheduling). The topic of scheduling exploits the following commands: **priority**, **bandwidth**, **bandwidth remaining**, **shape** and **fair-queue**. Using these commands we can apportion bandwidth when congestion exists and ensure that applications receive the treatment they need to operate over the network.

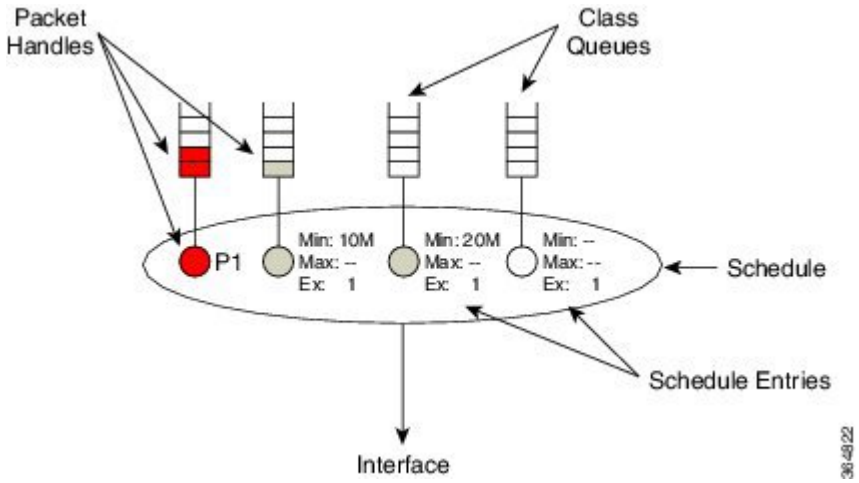
Specifically, this chapter will focus on flat policies attached to physical interfaces. The information presented here should ground your understanding of hierarchical scheduling concepts discussed in int following chapters.

About QoS Scheduling

Definitions

In this section we define core "scheduling" terms.

Figure 1: Scheduling Definitions



Packet Handle

When a router is prepared to forward a packet, it places a *packet handle*, representing that packet, in one of the egress queues. This handle holds information like the length of the packet and the location of the packet in memory.

Class Queues

When egress QoS is configured, a *class queue* is created for each class where we configure a queuing action. Similarly, we create an *implicit class-default queue* for any traffic not matching one of the explicitly-created queuing classes. If you configure a class with only non-queuing actions (e.g., a class with only marking configured), "matching" packets will be enqueued in the class-default queue.

Schedule

You should view a *schedule* (scheduler) as the decision maker. By selecting the packet handle, the schedule chooses which packet should next exit and when to send it. In the diagram above the "oval" represents a single schedule that selects a packet from one of the class queues.

Note An individual schedule is created for each interface.

Schedule Entry

For a schedule to choose between queues it needs to know each queues' expected treatment. We store this type of information in a *schedule entry*. For example, by configuring a queuing command (e.g. **bandwidth 10 Mbps**) you are setting the schedule entry.

The schedule entry also stores the internal state like the last time a packet was transmitted from that queue and the current packet handle, if any, from that queue.

Two types of schedule entries include the following: Priority Queues, on page 42, and Bandwidth Queues, on page 50.

How Schedule Entries are Programmed

In this section we provide a brief introduction to the parameters that are configured within a schedule entry. The actual commands will be covered in greater detail later in this chapter.

Firstly, a schedule entry is configured as either a *priority entry* or *bandwidth entry* (*priority queue* or *bandwidth queue*).

In the descriptions that follow you will see that priority entries can be further divided into *P1 entries* or *P2 entries*. You configure a P1 entry (the default) with either the **priority** or **priority level 1** command. Similarly, you use the **priority level 2** command to configure a P2 entry.

A bandwidth entry has three distinct parameters: *minimum rate* (Min), *maximum rate* (Max) and *excess weight* (drawn as "Ex" in illustrations).

Note The scheduler for a ASR 1000 Series Aggregation Services Router is often described as a three-parameter scheduler.

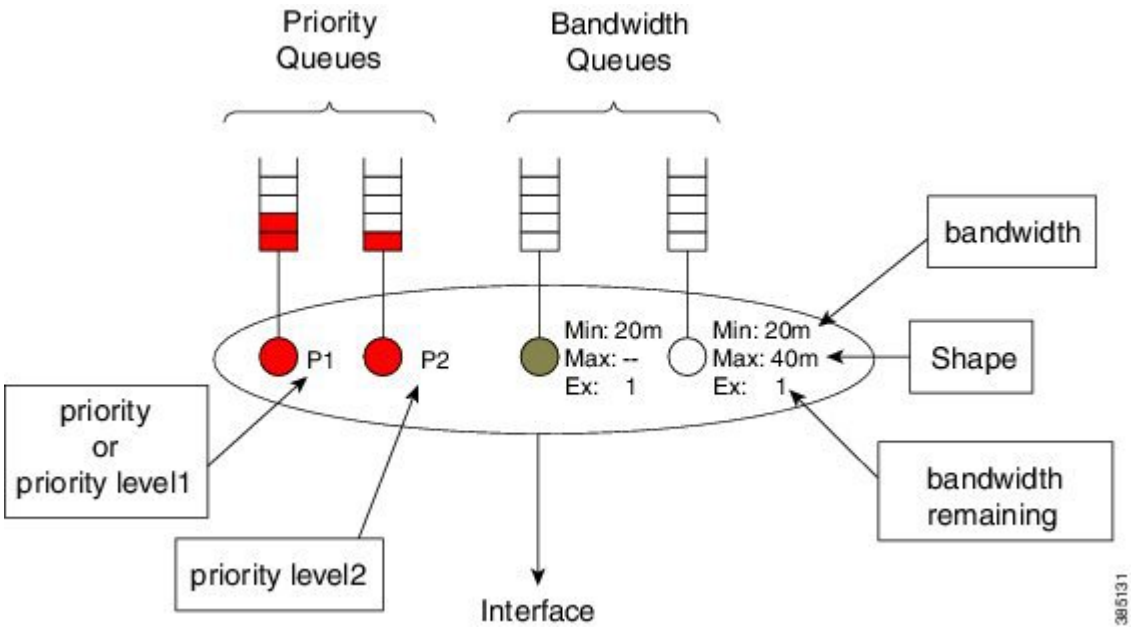
The **Min (minimum rate)** entry allocates a minimum bvandwidth guaranteed amount of throughput to a queue. The Min entry is configured with the **bandwidth** command and is not set unless explicitly configured. IOS configuration checking attempts to ensure that a schedule will always have sufficient bandwidth to honor any configured Min rates. Servicing queues based on monitoring throughput vs. a preconfigured target rate is sometimes referred to as *real time scheduling* (refer to Scheduler's Representation of Time, on page 59).

The **Max (maximum rate)** entry establishes a ceiling on the amount of throughput a queue can receive. The Max entry is configured using the **shape** command and is not set unless explicitly configured. Understand that Max sets a ceiling on the throughput of a queue but does not in itself guarantee any throughput to that queue.

The **Ex (excess weight)** entry mandates how queues will compete for any bandwidth available after Priority and Min guarantees have been met (*excess bandwidth*, or available bandwidth that is not guaranteed to, or not used by, priority and bandwidth guarantees). We configure Excess Weight with the **bandwidth remaining** command and unless explicitly configured, it defaults to 1. Excess *bandwidth sharing* is proportional to a queue's Excess Weight (sometimes referred to as *virtual time scheduling*, because no rates are configured and relative behavior alone is significant). For reflections on bandwidth sharing, see How Schedule Entries are Programmed, on page 33.

The following diagram summarizes what is presented above (the commands to set each schedule entry).

Figure 2: IOS Commands to Set Schedule Entries



Schedule Operation

How a schedule determines the packet sequence may be summarized as follows:

Note After each packet is forwarded, we return to step 1.

1. If the P1 queue is not empty, send the P1 packets.
2. If the PI queue is empty but the P2 queue is not, send the P2 packets.
3. Provided all priority queues are empty, the schedule services any queues with a minimum bandwidth guarantee (Min) and continues to service such queues until the guarantees are met. To ensure fairness, the scheduler will pick between queues with minimum guarantees by selecting the eligible queue, a queue that has not exceeded the bandwidth guarantee and has been waiting longest.
4. What if priority queues are empty and all bandwidth guarantees have been satisfied? Any excess bandwidth is distributed between queues that still require service until either all bandwidth is exhausted or a given queue has reached a maximum configured bandwidth. The Ex configured in that queue's schedule entry, dictates the share each queue will receive of this excess bandwidth.

Schedule Operation: Without a Shaper

The following example illustrates how a schedule operates and how it determines the bandwidth each queue will receive for a given offered load.

Before diving into the example we need to introduce the concept of *priority queue admission control*. In the previous description of schedule operation you will notice an absence of rates regarding how the schedule deals with priority queues; the schedule simply selects the priority queue whenever it contains a packet.

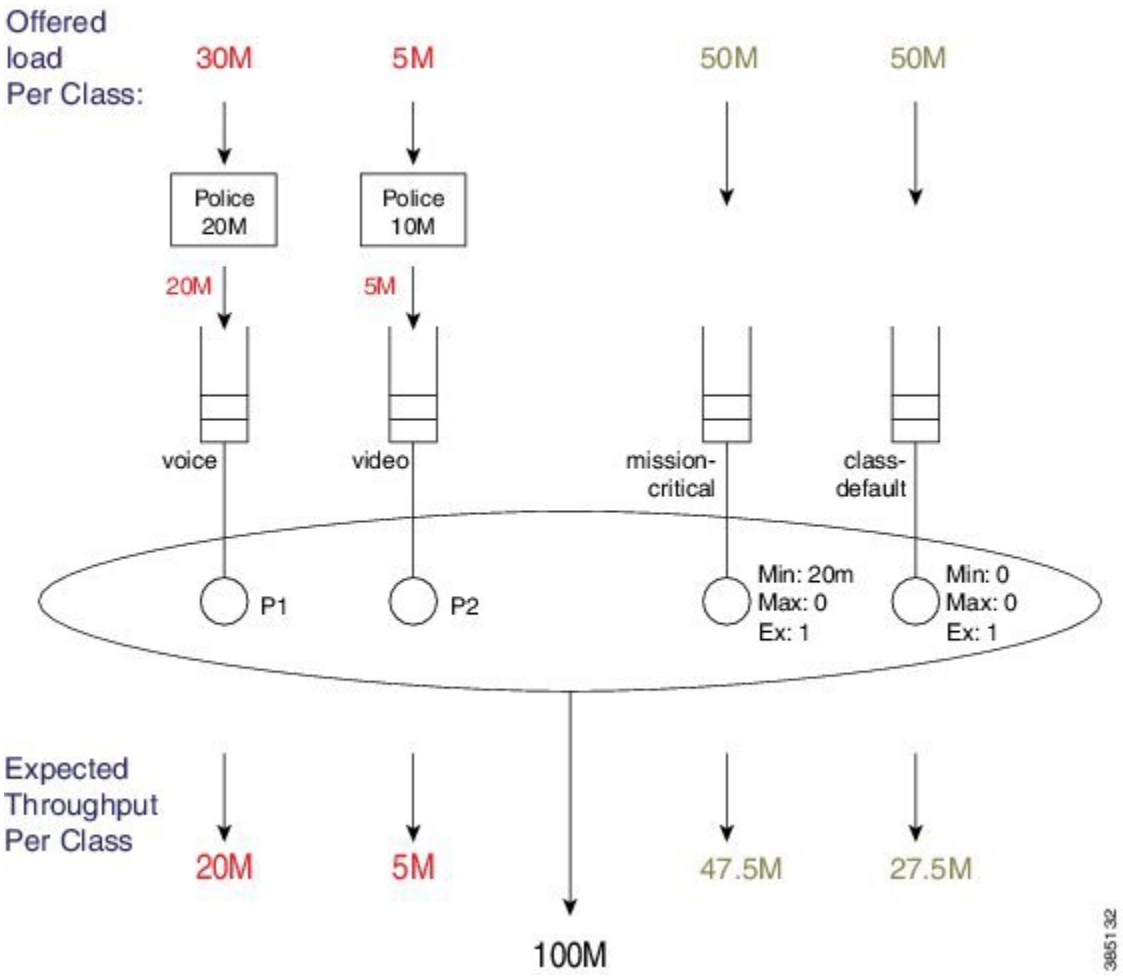
To prevent a priority queue (class) from starving other queues of service, we can use a policer to limit the consumable bandwidth. Such a policer restricts the rate at which packets can be enqueued in that queue.

In the following example, we attach a policy to a 100 Mbps interface:

```
policy-map scheduling-example
  class voice
    priority level 1
    police 20m
  class video
    priority level 2
    police 10m
  class mission-critical
    bandwidth 20000
  class class-default
```

Note Bandwidth is configured in Kbps. While **police** and **shape** commands support a postfix to specify the unit, the **bandwidth** command does not.

Figure 3: Scheduling Operation



The loads offered to each class are shown at the top of the figure: 30M, 5M, 50M, and 50M. We have applied policers (20M and 10M) to the priority queues.

30 Mbps is offered to the voice class, which first traverses a 20 Mbps policer, enqueuing 20 Mbps to the P1 queue. Because we always service this queue first, all 20 Mbps enqueued will be forwarded.

5 Mbps is offered to the video class (which all transits the 10 Mbps policer) and 5 Mbps is enqueued to the video queue. As 80 Mbps (100 Mbps - 20 Mbps) bandwidth is still available, all 5 Mbps will be forwarded.

After servicing priority queues, we advance to any queues with an explicit Min bandwidth guarantee. The mission-critical class has a Min of 20 Mbps so it will receive at least that amount of throughput.

The available excess bandwidth is 55Mbps (100 Mbps - 20 Mbps - 5 Mbps - 20 Mbps). Both the class-default and mission-critical classes have default excess weights of 1, so each receives an equal share of the available bandwidth, (55Mbps/2 =) 27.5 Mbps.

The mission-critical class will observe a total throughput of 47.5 Mbps (20 Mbps + 27.5 Mbps).

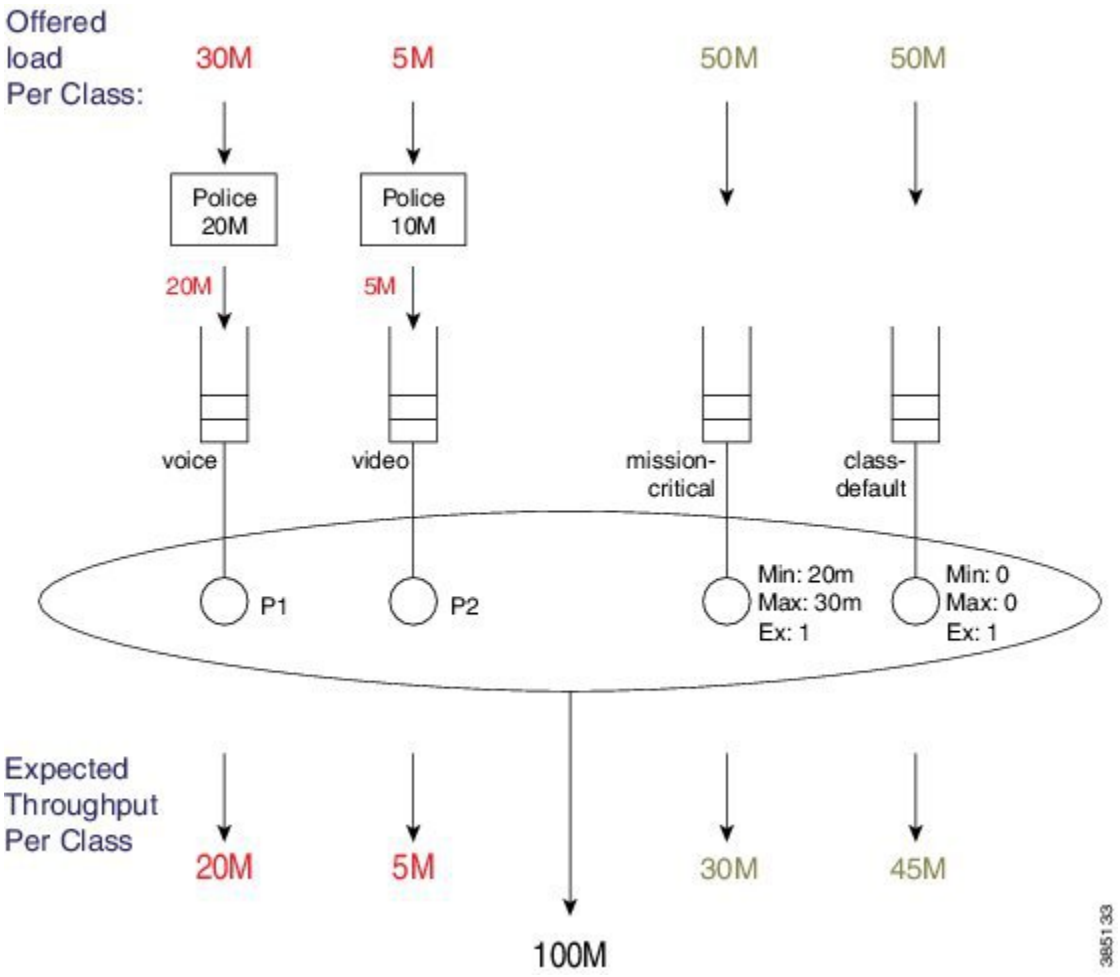
Schedule Operation: With a Shaper

Let's modify the configuration slightly - we will add a Max value (configure a shaper) to the mission-critical class:

```
policy-map scheduling-example
  class voice
    priority level 1
    police 20m
  class video
    priority level 2
    police 10m
  class mission-critical
    bandwidth 20000
    shape average 30m
```

Note We excluded class-default in the policy definition - it is always there whether or not we explicitly define it.

Figure 4: Scheduling Operation



The loads offered to each class is exactly as before: 30M, 5M, 50M and 50M.

30 Mbps is offered to the voice class, which first passes through a 20 Mbps policer, enqueing 20 Mbps to the P1 queue. We always service this queue first, so all 20 Mbps enqueued will be forwarded.

5 Mbps is offered to the video class (which all passes through the 10 Mbps policer) and 5 Mbps is enqueued to the P2 queue. As 80 Mbps (100 Mbps - 20 Mbps) bandwidth is still available, all 5 Mbps will be forwarded.

After servicing priority queues, we advance to any queues with an explicit Min. The mission-critical class has a bandwidth guarantee of 20 Mbps so it will receive at least that amount of throughput.

The available excess bandwidth is 55 Mbps (100 - 20 - 5 - 20 Mbps). Both the class-default and mission-critical classes have default Ex's 1, so each receives an equal share of the available bandwidth. From the Excess bandwidth sharing "rule," where in bandwidth is proportional to a queue's Ex, each class receives a 27.5 Mbps share. (For more information on this "rule," refer to How Schedule Entries are Programmed, on page 33.)

Based on the bandwidth guarantee and bandwidth sharing, the mission-critical queue would receive 47.5 Mbps (20 + 27.5 Mbps). However, the queue cannot use this much bandwidth because the Max configured shape rate is set to 30 Mbps (recall that Max is set to 0 in the previous example). Consequently, the queue uses 30 Mbps (out of the 47.5 Mbps received from bandwidth sharing) and the additional 17.5 Mbps of bandwidth returns to the excess pool.

As class-default is the only queue still requesting bandwidth, it has no competition and can consume this extra 17.5 Mbps, increasing its total throughput to 45 Mbps.

Note This example demonstrates how bandwidth is never wasted - scheduling will continue to sort through eligible queues and apportion bandwidth until one of the following applies:

- Each queue is empty.
- All Max values have been reached.
- All bandwidth has been consumed.

Configuring Rates and Burst Parameters

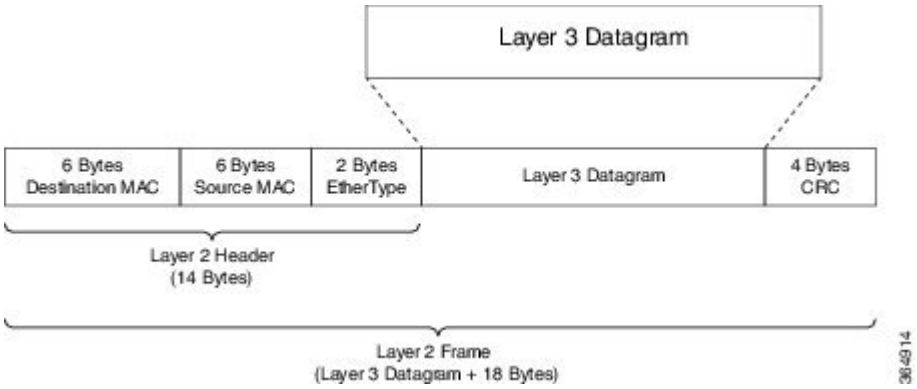
What's Included in Scheduling Rate Calculations (Overhead Accounting)

In the discussion of Schedule Operation, on page 34, you will notice that Min and Max are configured in bits per second. But what do these rates include? The short answer is that a schedule includes the Layer 3 datagram and Layer 2 header lengths but neither CRC nor inter-packet overhead.

Layer 3 Datagram

To clarify, let's imagine transporting an IP datagram over a GigabitEthernet link.

Note Henceforward, we will refer to a "schedule's perception of the packet length" as the *scheduling length*.

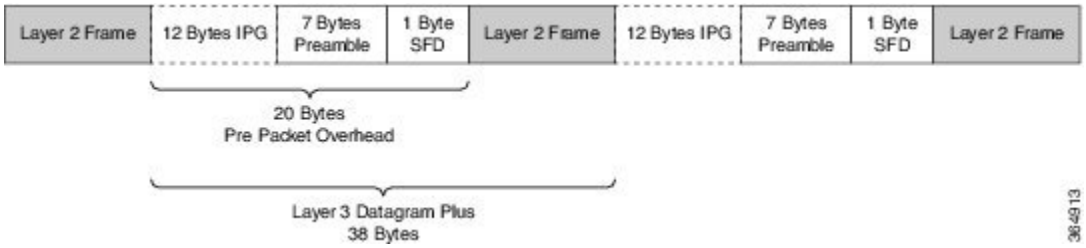


Ethernet Overhead

To transport the datagram on a GigabitEthernet link we first need to encapsulate it correctly in an Ethernet frame. This process adds 14 bytes of Layer 2 header and an additional 4 bytes of CRC (i.e., total of 18 bytes for encapsulation).

Consider what happens when this Layer 2 frame is transmitted over the physical medium. Ethernet requires a minimum *inter packet gap* (IPG) equal to the transmit time for 12 bytes of data, 7 bytes of preamble, and a *single-byte start-of-frame delimiter* (SFD), for a total pre-packet overhead of 20 bytes:

Figure 5: Ethernet Overhead



So, if you send multiple Ethernet frames sequentially, the total per-packet overhead for each Layer 3 datagram is an additional 38 bytes (encapsulation (18 bytes) + Ethernet inter-packet overhead (20 bytes)). For example, if you were to send 100 byte IP datagrams at line rate on a GigabitEthernet link, the expected throughput in packets per second would be:

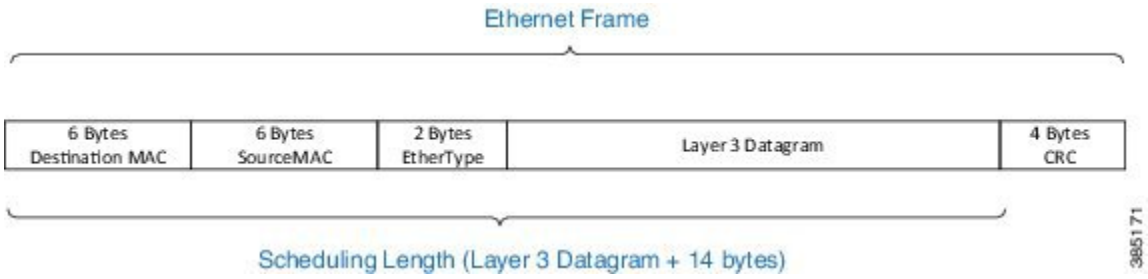
```
Linerate / Bits Per Byte / ( Layer 3 length + Per Packet Overhead) = Packets Per Second
```

```
1 Gbps / 8 / ( 100 + 38 ) = 905, 797 pps
```

Scheduling Length

From the scheduler's viewpoint, the packet's length is Layer 3 datagram + Layer 2 header (14 bytes on a GigabitEthernet interface):

Figure 6: Scheduling Length



Now consider a 500-Mbps shaper configured on a GigabitEthernet interface. (Recall that shaping is the process of imposing a maximum rate of traffic while regulating the traffic rate in such a way that downstream devices are not subjected to congestion.) As in the previous example, we will send all 100-byte IP datagrams to the scheduler, resulting in a "scheduling length" of 114 bytes (100 byte (datagram) + 14 byte (Ethernet Layer 2 header)). According to the following formula, the anticipated throughput would now be:

```
  Shaper Rate / Bits per Byte / (Layer 3 length + Layer 2 header length)
= Packets Per Second
```

```
  500 Mbps / 8 / (100 + 14) = 548,246 pps
```

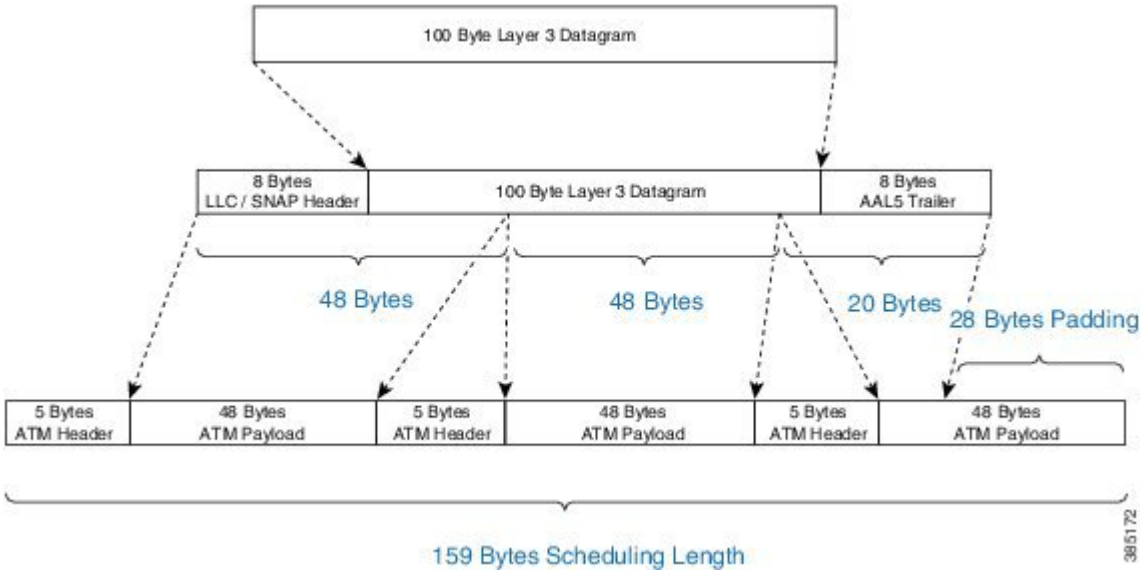
Observe that 100% of linerate (all 100 byte datagrams) was 905,797 packets per second but shaping to 500 Mbps (all 100 byte datagrams) yielded a throughput of 548,246 packets per second. Obviously, this is considerably more than 50% of physical capacity. When specifying rates to apportion bandwidth, be aware that rates do not include all overhead required to transport that packet.

Scheduler on an ATM Interface

When a queuing policy (scheduling policy) is attached to an ATM VC, scheduling rates in that policy are inclusive of all cell tax. This differs from a policer configured in such a policy that only includes the AAL5 header.

For example, consider a 100-byte datagram sent over an ATM VC configured with AAL5 SNAP encapsulation. A router will add an 8-byte LLC/SNAP header to the datagram, yielding a *policing length* of 108 bytes (analogous to a scheduling length of 114 bytes for a 100 byte IP datagram. (See What's Included in Scheduling Rate Calculations (Overhead Accounting), on page 38.) (For further details on policing length, refer to Priority Policing Length, on page 48.)

Figure 7: Scheduler on ATM Interface



To convey the packet, the router must also add an 8-byte AAL5 trailer (to the policing length) and then spilt the packet into ATM cells. To transport this packet we require 3 ATM cells, each carrying 48 bytes of the packet. We pad the third cell such that it also has a 48-byte payload.

Each of these 3 cells is 53 bytes in length (48 bytes packet + 5 byte ATM Header), which means that the scheduling length of the 100-byte datagram would be 159 bytes (3 cells x 53 bytes per cell).

Scheduler on a Logical Interface

In the Policing chapter we discuss how policer overhead accounting may differ depending on whether the policy is attached to a physical or logical interface (see Policer on Logical Interface, on page 268). This situation does not apply to a scheduler. Although a policy is attached to a logical interface (a tunnel interface) we must complete all processing and add any necessary headers before we enqueue the packet to egress a physical interface. Because we know the final length of the packet at the time of enqueue, we can set the scheduling length accordingly at that time.

Scheduler Overhead Accounting Adjustment

In prior sections, we described what is included by default in scheduler rate calculations. Occasionally, however, a user might want behavior to differ from the default.

For example, we hear that users want to express rates as physical bandwidth that would be consumed on the link. For an Ethernet interface, you would need to include the 4-byte CRC and 20-bytes inter-packet overhead required by each packet.

We also hear from service providers who want to charge their customers for traffic throughput at Layer 3 rates. The datagram's length remains constant as a packet traverses different interface types or encapsulating protocols, making it easier for users to understand. In this instance, we would not include the Layer 2 header length in shape rate calculations.

Note Changing overhead accounting may impact the network elsewhere. For example, if we use a policer for network admission control, we typically configure a shaper on customer premises equipment connecting to that network. The shaper and policer should have the same view of what is included in CIR.

Scheduler Account Option

The scheduler account option (the **account** keyword) allows you to specify a number of bytes that should be added or removed from the default "scheduling length" per packet to achieve the desired behavior. You can add or subtract at most 63 bytes per packet. This option is supported on the **shape** and **bandwidth** commands.

In the following example, we apply a shaper on an Ethernet interface and we want to include all overhead such that the shaper will cap throughput at 50% of the actual physical bandwidth. By adding 24 bytes per packet we "cover" the 4 byte CRC and 20-bytes inter-packet overhead:

```
policy-map ethernet-physical-example
  class class-default
    shape average percent 50 account user-defined 24
```

Note With overhead accounting, we must account for *hierarchical policies*. If a parent shaper is configured with the account option, any child shapers or bandwidth guarantees will also inherit the same adjustment as specified in the parent policy.

In the chapter on hierarchical scheduling, we will observe how to use shapers to condition traffic for remote links and to use child polices for apportioning bandwidth within that shape rate. In that use case, the encapsulation on the remote link may differ from the encapsulation on the sending device (e.g. an enterprise hub router connected to the network with an Ethernet interface sends traffic to a branch connected with a T1 interface). If the T1 link were using HDLC encapsulation each datagram would have 4 bytes of Layer 2 headers on that link. On the Ethernet, however, each packet would have 14 bytes of Layer 2 headers. The account option can be used to shape and schedule packets as they would appear on that remote link. That is, remove 14 bytes from the scheduling length as Ethernet headers are no longer present and then add 4 bytes to the scheduling length to represent the HDLC Layer 2 overhead.

Overhead Accounting Adjustment (Predefined Options)

In addition to specifying a number of bytes to add or subtract (see the following table), the CLI also offers some predefined options with which you can specify remote encapsulation. The current predefined options are based on broadband use cases, assuming that we send (or receive) traffic on an Ethernet interface to a DSLAM elsewhere in the network. Although we are encapsulating in Ethernet frames that include Dot1Q or Q-in-Q, the DSLAM receives some form of ATM encapsulation. We want the shaper to condition traffic to mirror how it would appear after DSLAM. In each case we would also add cell-tax to the scheduling length.

Imagine that we forwarding Dot1Q encapsulated packets on an Ethernet interface. Imagine further than a downstream DSLAM will:

- receive the packets
- strip the Ethernet and Dot1q headers
- perform AAL5-Mux 1483 routed encapsulation.

Referring to the previous table, DSLAM will remove 18 bytes of Ethernet/Dot1q and add a 3-byte LLC header, generating a -3 bytes change in the scheduling length. (For a schematic, refer to What's Included in Scheduling Rate Calculations (Overhead Accounting), on page 38.)

As the DSLAM is sending over an ATM network, it would add an 8-byte AAL trailer and then split the resulting PDU into 53-byte cells. The ATM value "yes" (with reference to the table) indicates that the router will calculate this cell tax and include that extra overhead in the scheduling length.

```
policy-map atm-example
  class class-default
    shape average 50m account dot1q aal5 mux-1483routed
```

Example - Predefined Overhead Accounting

Imagine we are forwarding Dot1Q encapsulated packets on an Ethernet interface. Imagine further than a downstream DSLAM will:

- receive the packets
- strip the Ethernet and Dot1q headers
- perform AAL5-Mux 1483 routed encapsulation.

Referring to the previous table, DSLAM will remove 18 bytes of Ethernet/Dot1q and add a 3-byte LLC header, generating a -3 bytes change in the scheduling length. (For a schematic, refer to What's Included in Scheduling Rate Calculations (Overhead Accounting), on page 38.)

As the DSLAM is sending over an ATM network, it would add an 8-byte AAL trailer and then split the resulting PDU into 53-byte cells. The ATM value "yes" (with reference to the table) indicates that the router will calculate this cell tax and include that extra overhead in the scheduling length.

```
policy-map atm-example
  class class-default
    shape average 50m account dot1q aal5 mux-1483routed
```

Priority Queues

Priority queues represents a type of schedule entries that enable you to avoid any unnecessary delay in forwarding packets. Through the priority semantic, we can guarantee low latency treatment for applications that are latency and (or) jitter sensitive. As an example, consider Voice over IP (VOIP). Typical VOIP phones have a 30 mS *de-jitter buffer*, allowing them to tolerate a maximum of 30 mS jitter end-to-end across the network.

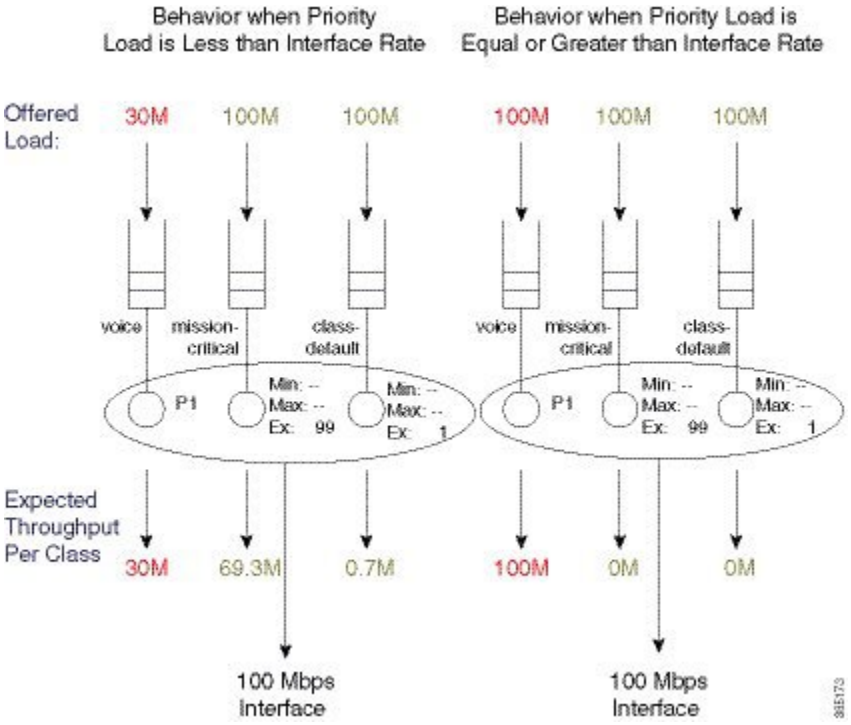
When you configure a priority queue, you can select one of three ways to control the bandwidth that might be consumed by that traffic class: unconstrained priority queue, conditional policer, or (un-conditional) always-on policer.

Unconstrained Priority Queue

One way to control bandwidth consumption is with an *unconstrained priority queue* (absolute priority queue), which is a priority queue configured without any limit on the amount of bandwidth that may be consumed by the priority class. To illustrate, consider this example configuration as well as the configured schedule entries in the following figure. Note the lack of a policer for admission control to the priority queue.

```
policy-map absolute_pq_example
  class voice
    priority
  class mission-critical
    bandwidth remaining percent 99
  class class-default
    bandwidth remaining percent 1
```

Figure 8: Unconstrained Priority Queue



In the example on the left, the actual interface bandwidth capacity (100 Mbps) exceeds the load to the priority queue of the voice class (30 Mbps). This leaves 70 Mbps of excess bandwidth to apportion based on the excess weight (Ex) ratio (99:1; set by the **bandwidth remaining** command).

In the example on the right, the priority load has been increased to 100 Mbps. Because this leaves no excess bandwidth, other queues are starved of service (an expected throughput of 0M).

The take-home is that without admission control on a priority class, a class might consume the entire interface bandwidth and so starve all other service queues. This can cause mission-critical applications to suffer. Moreover, if control messages are in the starved service queue, network instability might result.

So, use unconstrained priority queues with caution. To ensure the priority queue is unable to starve others of service, you might want to consider using alternative bandwidth control systems like Call Admission Control (CAC).

Note You cannot use minimum bandwidth guarantees (as set by the **bandwidth** command) in conjunction with unconstrained priority queues. If the priority queue is capable of consuming all available bandwidth it follows that you can't guarantee any of that bandwidth to other classes. IOS will reject any such configurations.

Priority Queue with Conditional Policer

Another way to control bandwidth consumption is to enter a value with the **priority** command. This represents a way to handle queue admission control with a conditional policer.

Conditional priority rate limits traffic with a policer only if congestion exists at the parent (policy-map or physical interface) level. This state exists provided more than the configured maximum rate of traffic attempts to move through the class (and/or interface).

The key element is that a conditional policer will only drop packets if the schedule is congested. That is, it will only drop packets when the offered load exceeds the available bandwidth (interface bandwidth in the context of a flat policy-map attached to a physical interface).

A conditional priority class can use more than its configured rate, but only if contention with other classes in the same policy is absent.

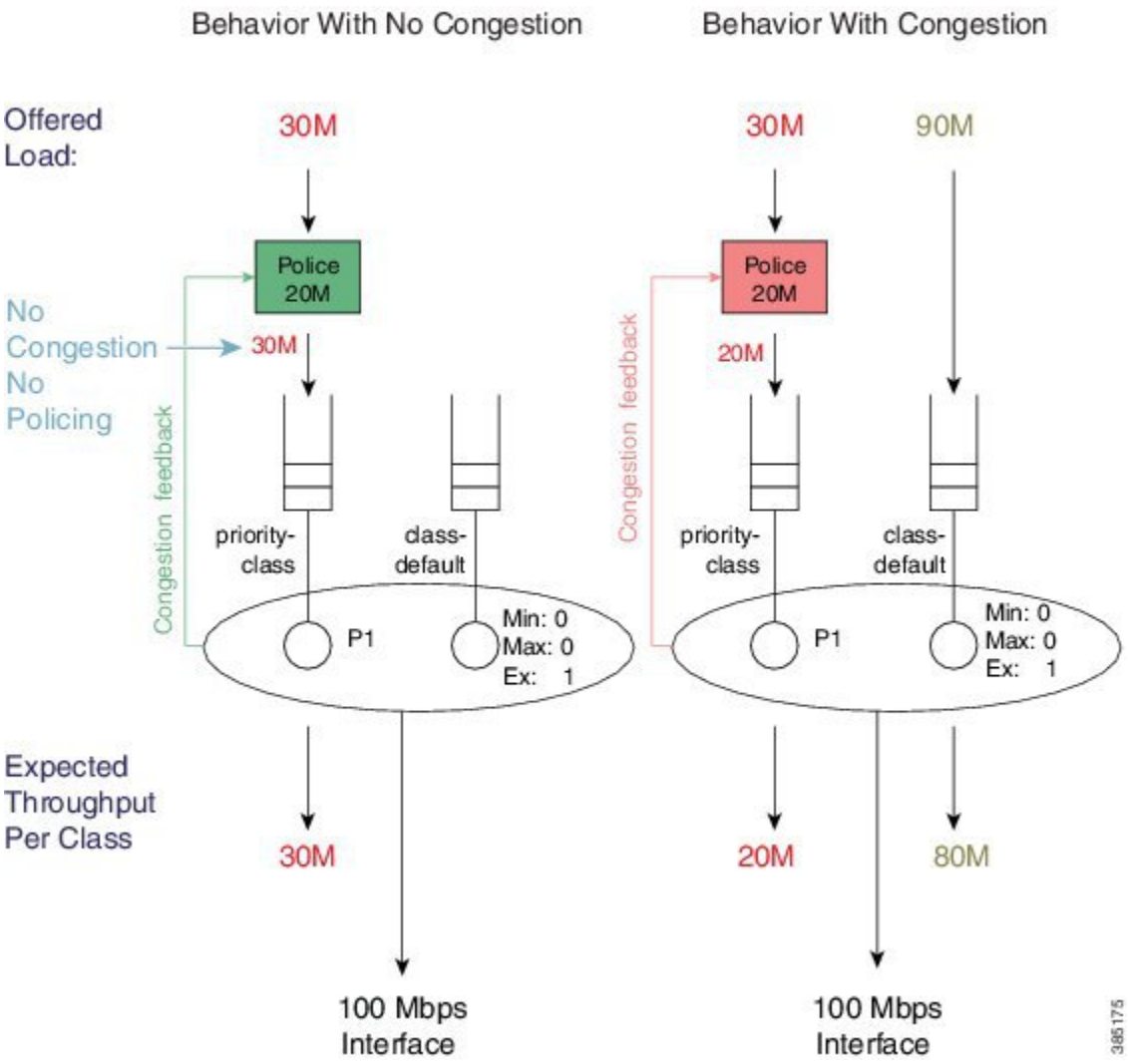
The schedule provides congestion feedback to the policer, which will count every packet in the rate it observes but will suppress the drop action unless congestion exists. So, if no congestion exists, the priority queue may consume whatever bandwidth is available but when congestion occurs the queue will be policed to the configured rate.

```
policy-map conditional_policer_example
  class priority-class
    priority 20000
```

The priority value is configured in kps and although configured with the **priority** command, it does not alter a schedule entry. (Recall that a schedule entry is where we store a queue's expected treatment.) Instead, it configures a policer that will be executed before packets may be enqueued.

The following diagram shows how a conditional policer would function with and without congestion. In this example, we have attached the previous configuration to a 100 Mbps interface.

Figure 9: Priority Queue with Conditional Policer



In the schedule depicted on the left no congestion exists. As the congestion feedback reports no congestion and the policer will enqueue the entire 30 Mbps offered to that class.

In the schedule depicted on the right where congestion exists, the policer will enforce the 20 Mbps rate configured with the **priority** command.

Be aware that a conditional policer has advantages and disadvantages:

Advantage	A priority class may use all bandwidth not currently used by other classes.

Disadvantage	You cannot carefully plan for priority capacity throughout your network if you don't know the forwarding-capacity of a particular interface. A *true priority service* should have low latency (no queue build-up) and no drops, end-to-end. You experience inconsistent behavior depending on whether or not an interface is congested. If you under-provision the police rate, you may observe intermittent problems with applications using that class and diagnosing the issue might be very difficult.

Note You cannot use conditional policers and policer overhead accounting adjustment concurrently.

Priority Queue with Always on (Unconditional) Policer

The third way to control bandwidth consumption is to use an explicit always-on (i.e., unconditional) policer for queue admission control.

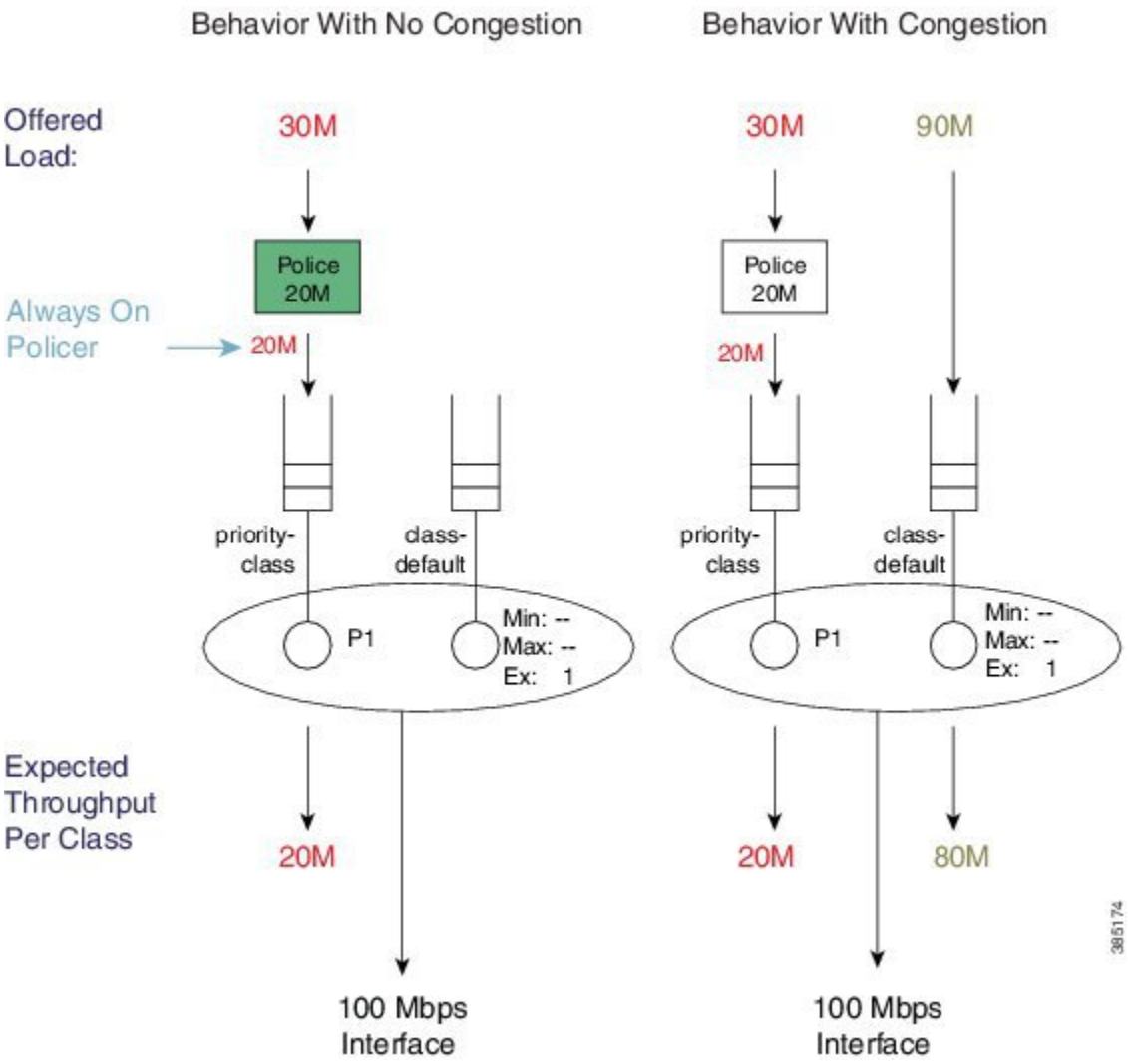
When you configure a priority class with an explicit policing rate, traffic is limited to the policer rate regardless of congestion conditions. That is, even if bandwith is available, the priority traffic cannot exceed the explicit rate.

The following example shows how such a configuration might look:

```
policy-map always_on_policer
  class priority-class
    priority
    police cir 20m
```

The diagram below shows the behavior of such a policer.

Figure 10: Always on Policer



When you configure a priority class with an explicit policing rate, this rate is always enforced. That is, even with sufficient bandwith, priority traffic cannot exceed the explicit rate. This means that you have *deterministic behavior* in your priority service. If a user complains of poor application performance you can look for policer drops in your network and determine if insufficient bandwidth is allocated to the priority service. Applications should have the same experience regardless of whether or not congestion exists.

Priority Queue Burst Considerations

In previous sections we describe how to perform queue admission control for the priority queue using a conditional or always-on policer. Specifically we employ a *single-rate two-color policer*. From the policing chapter we know that policers are implemented using a token bucket scheme that allows for some burst (Single-Rate, Two-Color Policer, on page 263). Controlling this (burst) allowance is crucial when you use policers in this way.

Bursts that are allowed by the policer may result in a build-up of the priority queue, which will generate latency for a packet that is added to the end of that queue. The packet must wait for the transmission of all

preceding packets in the queue before it too can be transmitted. The amount of latency depends on the *serialization delay*, the time taken to transmit those packets on the physical medium.

As multiple packets from a given flow arrive, they may experience different conditions in the priority queue. One might arrive and be enqueued behind a number of packets already waiting and thus experience some latency. The next packet in that same flow may arrive to an empty priority queue and be scheduled immediately. What this means is that any potential latency from priority queue congestion is also potential *jitter* (jitter, potential the variation in the latency of received packets).

The *default burst allowance* for policers in IOS is set at 250 mS for always on policers and 200 mS for conditional policers. If a policer can allow us to enqueue a burst, it follows that these numbers can be almost directly translated into potential jitter. In the introduction to the priority semantic (see Priority Queues, on page 42), we indicated that voice applications can typically tolerate about 30 mS of jitter and 150 mS latency end-to-end across a network. Given the former, we usually try to apportion some of this budget to each node in the network. A simple guideline is to allow a burst tolerance (and thereby potential jitter) of 5-10 mS on any single node.

For example, envisage a priority queue configured with a queue-admission policer at a rate of 2 Mbps and a burst allowance of 5 mS. Calculate the number of bytes we can transmit in 5 mS:

Burst Target
= Police Rate / 8 Bites per Byte * 5 mS
= 2 Mbps / 8 * .005 = 1250 bytes

For an always on policer, the configuration for this example would look like:

```
policy-map always_on_policer_burst_example
  class voice
  priority
  police cir 2000000 1250
```

For a conditional policer, the configuration example would look like:

```
policy-map conditional_policer_burst_example
  class voice
  priority 20000 1250
```

Priority Policing Length

In the section What's Included in Scheduling Rate Calculations (Overhead Accounting), on page 38 we introduced the concept of scheduling length, which is how a scheduler "views" packet length when it is evaluating conformance to a rate. In the Policing chapter we also introduced the similar concept of policing length (What's Included in the Policer-Rate Calculation (Overhead Accounting), on page 267). As the rate configured on a priority queue is a policing rate, we will use the policing length when determining conformance to that rate. When a policy is attached to a physical interface, as described in this chapter, the policing and scheduling lengths are identical. To alter the policing length, you can use the policer overhead accounting feature.

Note The **account** keyword is supported with always-on policers but not conditional policers.

Multi-Level Priority Queuing

The Multi-Level Priority Queues (MPQ) feature allows you to configure multiple priority queues for multiple traffic classes by specifying a different priority level for each of the traffic classes in a single service policy-map.

In Schedule Operation, on page 34, we introduced that a priority queue may be P1 or P2. The original intent of this feature was to support voice and video in separate priority queues as each has differing traffic characteristics and jitter tolerance. In particular, voice has a smaller packet size (typically around 80 bytes for voice compared to 1400 bytes for video) and tighter jitter requirements (typically 30 mS whereas non-interactive video may be 100's of mS). So, we would use a P1 queue for voice and a P2 queue for video traffic.

Today many video applications use advanced adaptive codecs and separate the voice and video content into separate streams. Some argue that video traffic is now TCP-like in its behavior and does better in bandwidth queues. Interactive video on slower links may still require a P2 queue.

To configure multilevel priority queuing you must use the **level** keyword in the **priority** command. This feature is supported with conditional policers, always-on policers and absolute priority queues.

Here is an example of a multilevel priority queue with conditional policers:

```
policy-map multilevel-example2
  class voice
    priority level 1 5000 3125
  class video
    priority level 2 10000 12500
```

Here is an example of a multilevel priority queue configuration with always-on policers:

```
policy-map multilevel-example1
  class voice
    priority level 1
    police cir 5000000 3125
  class video
    priority level 2
    police cir 10000000 12500
```

Note If you do not explicitly configure a level, a priority queue will operate as a P1 queue. However, if you want to configure multilevel priority queuing, you must explicitly configure levels.

For example, the following configuration would be rejected - you need to explicitly configure the priority level in the voice class:

```
policy-map multilevel-rejection-example
  class voice
    priority
    police cir 5000000 3125
  class video
    priority level 2
    police cir 10000000 12500
```

Bandwidth Queues

Bandwidth queues enable you to apportion interface bandwidth for applications that lack strict latency requirements. Recall that the intent of scheduling is to ensure that all applications receive the necessary bandwidth and to utilize unused bandwidth by making it available for other applications.

You can reflect on *bandwidth sharing* as follows:

Guarantee bandwidth for applications so that they operate effectively. For example, you may decide that your email application is business critical and must continue to operate even during network congestion. If so, you would want to always guarantee some amount of available bandwidth to your business critical applications.

Determine which applications to sacrifice under congestion. For instance, you may decide that a social media application is not business critical; employees can use the network for such applications but not at the expense of business critical activities. If so, you can place these applications in a queue that is intentionally deprived of service during congestion.

As described in How Schedule Entries are Programmed, on page 33, bandwidth queue schedule entries have three distinct parameters Min, Max, and Ex set by **bandwidth**, **shape**, and **bandwidth remaining** commands, respectively. Let's take a closer look at these commands.

Bandwidth Command

The **bandwidth** command sets the Minimum bandwidth (Min) guarantee in a schedule entry at which a queue will be serviced. Given the exact bandwidth requirements of an application, this command provides a convenient way to ensure that an application receives exactly what it needs under congestion. Be aware that by default every entry will also have a configured Excess Weight, which can lead to some additional guaranteed service for the queue.

Bandwidth guarantees are configured in Kbit/sec and may be configured in increments of 1 Kbps. You can also configure the guarantee as a percentage of physical linerate: a percentage of the nominal interface rate displayed as bandwidth through the **show interface** command.

For example, the **show interface gigabit x/y/z** command on a GigabitEthernet interface would show a BW of 1000000 Kbit/sec and any percentage value would be a percentage of this nominal rate. So, if we configured **bandwidth percent 50** on a Gigabitethernet interface, it would set a Min value of 500 Mbps.

The **bandwidth** command accepts the **account** keyword, which enables you to adjust what overhead is included in rate conformance calculations. However, any configured **account** value must be consistent across a policy-map (all **bandwidth** and all **shape** commands in the policy-map must be configured with the same account value).

Caution Do no configure extremely low bandwidth guarantees on high speed interfaces.

Recall from How Schedule Entries are Programmed, on page 33, that we service queues with Min bandwidth guarantees before Excess queues. Furthermore, recall from What's Included in Scheduling Rate Calculations (Overhead Accounting), on page 38 that by default the scheduling length of a packet does not include all of the physical bandwidth that will be consumed to transport that packet (it does not include CRC or inter packet overhead). Given these two facts you should be careful not to guarantee more bandwidth than is actually available. Else, you may starve queues possessing only an excess weight of service.

Imagine that you configure a queue with a Min of 98 Mbps and attach the policy to a FastEthernet interface (100 Mbps). If we send all 100 byte frames, the *scheduling length* for each frame would be 96 bytes but the *actual bandwidth* consumed by each (including the required inter- packet overhead) would be 120 bytes.

From a scheduling length perspective, 98 Mbs would translate to 120 Bytes/96 Bytes * 98 Mbps = 122.5 Mbps of physical bandwidth usage:

actual bandwidth/scheduling length * Min = physical bandwidth usage

(100 bytes Frame + 20 bytes Per Packet Overhead)/(100 bytes Frame - 4 bytes CRC) * 98 Mbps = 120 bytes/96 bytes * 98 Mbps = 122.5 Mbps

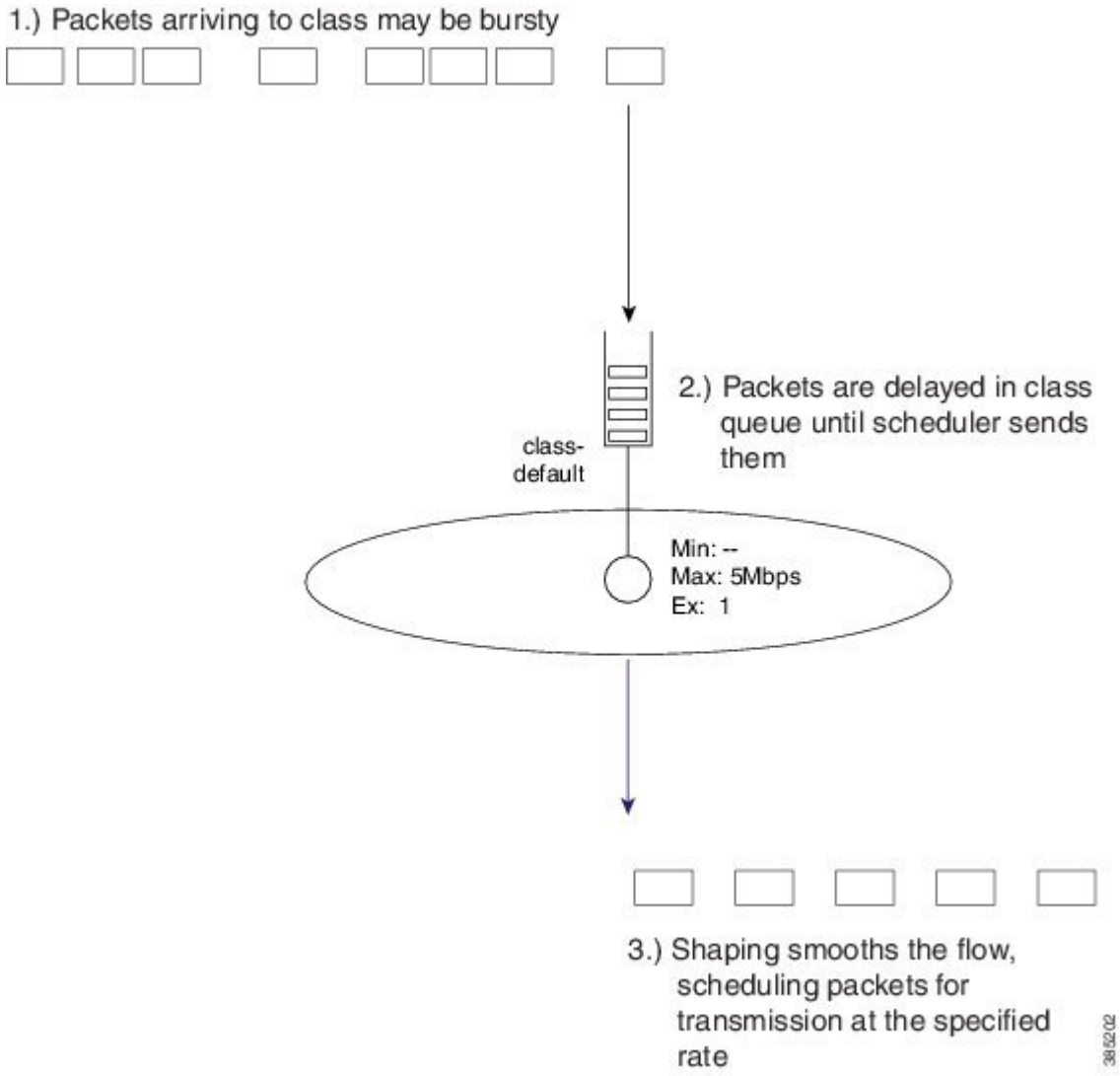
So, we cannot honor our promise! Generally, if the sum of your priority guarantees and Min bandwidth guarantees totals 75% or more of the physical bandwidth, consider whether you are starving other queues of service. In particular consider what might happen to class-default traffic as the default configuration for the class-default schedule entry is to configure an excess weight only.

Shape Command

The **shape** command sets the Max rate in a schedule entry at which a queue will be serviced. Setting the Max rate does not in itself guarantee any throughput to that queue; it simply sets a ceiling. If you create a class containing just the **shape** command it will also receive the default excess weight setting (‘1'), which determines the bandwidth share that class should receive.

Shaping is most commonly used in hierarchical policies. Occasionally, however, you might want to use shaping in flat policies. That is, you may have adaptive video in a bandwidth class that if unconstrained could expand its bandwidth usage to beyond what is physically available. Consequently, you might want to employ shaping to limit the expansion of that flow.

Figure 11: Single-shaped Queue



The diagram above shows an example of a single-shaped queue. For this simple example, the configuration would look as follows:

```
policy-map shape_example
  class class-default
    shape average 5m
```

As packets arrive they are added to the end of the queue for that class. The scheduler is pulling packets from the head of the queue at the specified rate. If the *arrival rate* (rate at which packets are arriving at the queue) exceeds the *service rate* (the rate at which packets are pulled from the queue) then packets will be delayed and must sit in the queue until all preceding packets are sent. In this simple example no other queues compete for bandwidth, so the service rate will equal the shape rate (5 Mbps).

From this simple example you can see that a shaper will "smooth" a stream. Typically, it will be a few small packets rather than a single packet released by the scheduler. The net result is as shown, a shaper meters the rate at which packets are forwarded.

Shape Average

The **shape average** command is the primary means of configuring a Max rate for a class.

You can configure the rate in bits per second or as a percentage of the interface (or parent shaper) rate. As with other scheduling commands, you can adjust the overhead included in scheduling calculations with the **account** keyword.

As an example, we can modify the previous configuration snippet to include CRC and inter-packet overhead on an Ethernet interface as follows (for more details see What's Included in Scheduling Rate Calculations (Overhead Accounting), on page 38):

```
policy-map shape_example
  class class-default
    shape average 5m account user-defined 24
```

Note The **shape average** command-line interface also includes options for Bc (bits per interval, sustained or committed) and Be (bits per interval, excess). (These options are remnants from the software implementation of shaping in IOS classic and have no effect on an ASR 1000 Series Aggregation Services Routers.)

On software implementations the processing overhead meant it was only feasible to perform the math involved in scheduling at some predetermined interval, typically a number of milliseconds.

Adjusting Bc was a way to further reduce scheduling frequency (and thereby processing overhead) at the expense of more burstiness in forwarded traffic. On the ASR 1000 Series Aggregation Services Routers, scheduling decisions are performed in dedicated hardware and (**so?**) frequent scheduling decisions does not incur a performance penalty. We have optimized the hardware to maximize the elimination of burstiness from the stream it forwards, obviating user input on Bc or Be.

Shape Peak

The **shape peak** command is supported on the ASR 1000 Series Aggregation Services Router but it offers no functionality beyond the **shape average** command. We support it to easily migrate configurations from existing IOS classic devices to ASR 1000 Series Aggregation Services Routers. With **shape peak** command, the router will look at the configured rate, Bc and Be and then calculate a target shape rate. This rate displays in the **show policy-map interface** command output and on the ASR 1000 Series Aggregation Services Router is programmed into the hardware schedule entry. If you are creating a new configuration, you should use the **shape average** command.

Bandwidth Remaining Command

The **bandwidth remaining** command configures the excess weight in a schedule entry and so determines a queues share of the excess bandwidth. Recall that excess bandwidth is defined as any bandwidth that is neither explicitly guaranteed to another queue by the **priority** or **bandwidth** command nor used by a queue to which it is guaranteed. (For details on excess weight, see How Schedule Entries are Programmed, on page 33.) By distributing excess bandwidth sharing in a deterministic manner (behavior entirely determined by initial state), we avoid wasting bandwidth. (For further discussion of bandwidth sharing, see Bandwidth Queues, on page 50.)

The **bandwidth remaining** command is also an effective way to guarantee bandwidth to queues. It is perfectly reasonable, and very common, to apportion all bandwidth using only excess bandwidth sharing.

The **bandwidth remaining** command has two variants - **bandwidth remaining ratio** and **bandwidth remaining percent**. in either case, you are setting the same excess bandwidth parameter in a schedule entry. The rationale for two forms will make sense when we discuss hierarchical policies. In the context of a flat policy attached to a physical interface, however, you can choose whichever form simplifies provisioning.

Note Both variants (as similar to other scheduling commands) support the **account** keyword.

Bandwidth Remaining Ratio

Concerning the **bandwidth remaining ratio** command, the first thing you need to understand is that every bandwidth queue schedule entry will have a default Excess Weight (Ex) of one ('1') provided you do not explicitly set the value. (For example, upon creation, the schedule entry for the class-default queue will have an Ex of 1.) Having a deterministic and easy to understand default removes any ambiguity when designing a QoS scheme.

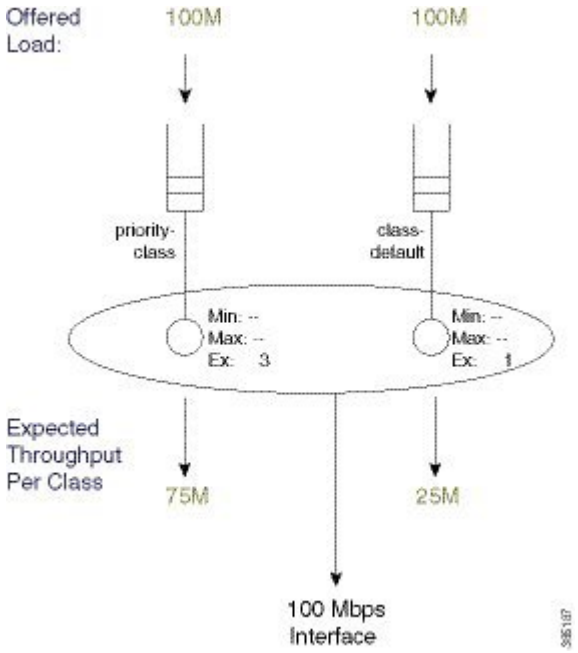
Consider the following policy-map example:

```
policy-map BRR-Example1
  class mission-critical
    bandwidth remaining ratio 3
```

This policy has 2 queues, one for the mission-critical class we explicitly create with a scheduling command and one for the implicit class-default.

Let's now attach this policy to a 100 Mbps interface and offer 100 Mbps to each queue. The scheduling hierarchy and expected throughput per class would be as shown:

Figure 12: Splitting Bandwidth Explicitly Assigned by Ratio



Now let's modify the policy by adding an explicit class with the **shape** command. Recall from the command page that the **shape peak** command sets the Max of the schedule entry for that queue. Because Ex has not been explicitly configured, it will default to 1.

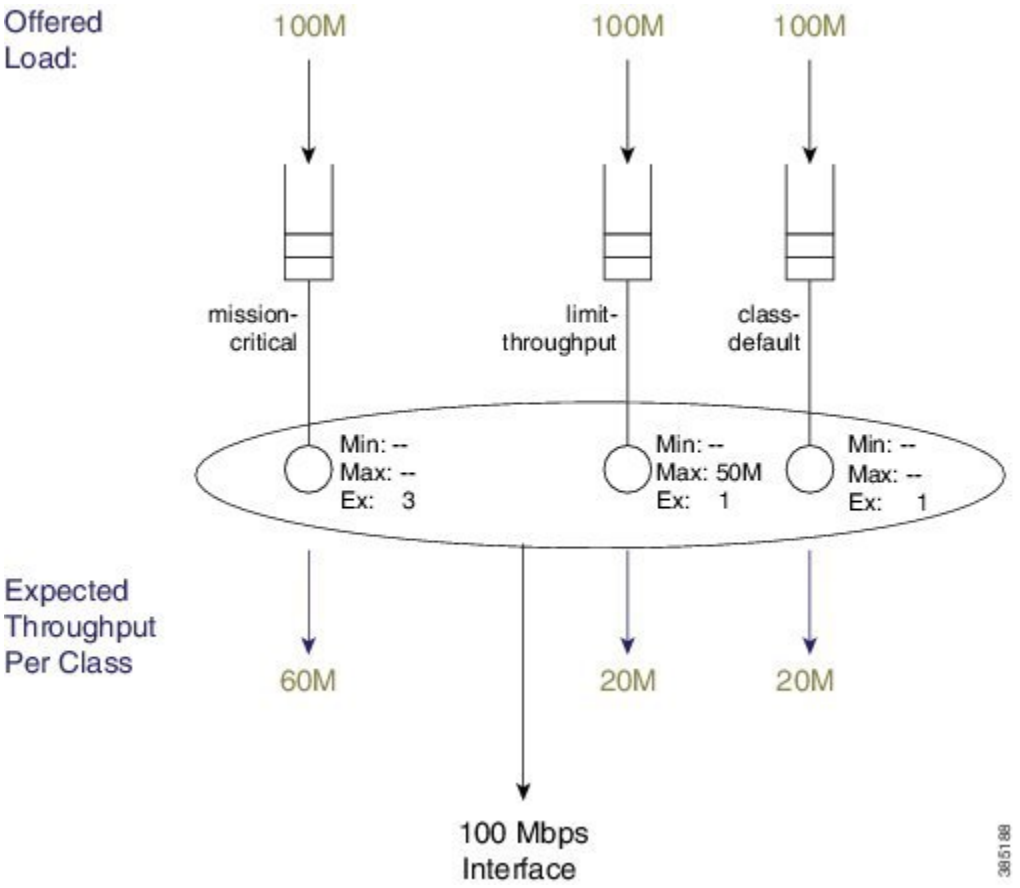
Note A Max entry does not guarantee any share of the bandwidth to a queue, it simply sets a ceiling on the possible throughput for that queue.

The policy could look like this:

```
policy-map BRR-Example1
  class mission-critical
    bandwidth remaining ratio 3
  class limit-throughput
    shape average 50m
```

If we attach this policy to a 100 Mbps interface and offer 100 Mbps to each class, the scheduling hierarchy and expected throughput would look as follows:

Figure 13: Modifying Excess Weight of Explicit Classes with bandwidth remaining ratio Command



The expected throughput (60M, 20M, and 20M) reflects the ratio of Ex values: 3, 1 , and 1. The key point is that modifying the excess weight using the **bandwidth remaining ratio** command will only alter the entry for the class you are explicitly modifying.

The bandwidth remaining ratio ranges from 1 to 1000 so we can achieve considerable variance between the service rate for different queues.

Bandwidth Remaining Percent

The **bandwidth remaining percent** command is another way to modify the Excess Weight (Ex) in a bandwidth queue's schedule entry. Obviously, with a percent-based scheme, the sum of excess weights across all bandwidth queues must total 100. We achieve this by distributing (equally) any percentage (not explicitly assigned) across class-default and any other queues that are not configured explicitly.

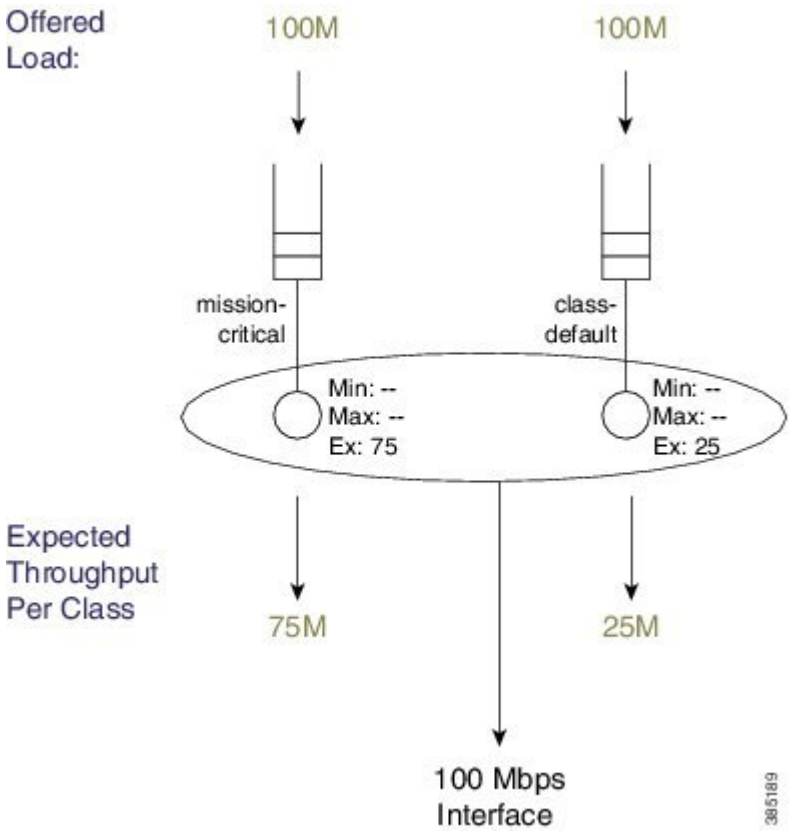
For details on this command, refer to the command page for bandwidth [remaining percent].

Consider the simplest example, which is equivalent to the first example in bandwidth remaining raito:

```
policy-map BRP-Example1
  class mission-critical
    bandwidth remaining percent 75
```

The scheduling hierarchy and expected throughput per class will look as follows:

Figure 14: Splitting Bandwidth Explicitly Assigned by Percent



Notice how the Ex of class-default was changed (from "1," by default) even though it was not explicitly configured.

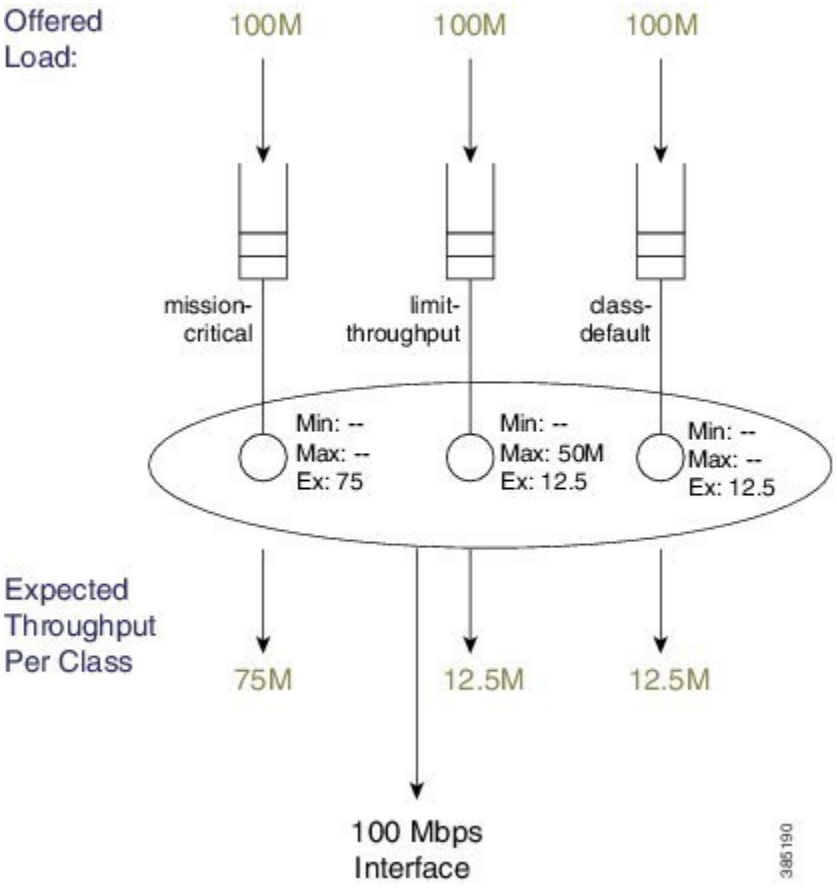
Now let's add a queuing class with no explicit bandwidth remaining configuration - again we'll add a class with just a shaper (see the figure "Splitting Bandwidth Explicitly Assigned by Ratio" in bandwidth remaining ratio:

```
policy-map BRP-Example2
  class mission-critical
    bandwidth remaining percent 75
  class limit-throughput
    shape average 50m
```

This example highlights the behavior of splitting percentage across class-default and any classes that are not explicitly assigned.

The hierarchy and throughput will now look as follows:

Figure 15: Splitting Bandwidth Percentage Across class-default and Unassigned Classes with an Added Shaper



Two-Parameter versus Three-Parameter Scheduling

Earlier we described how the schedule entry for each bandwidth queue has three parameters to control queue service: Min, Max and Ex. (See How Schedule Entries are Programmed, on page 33.) This is why we categorize the scheduler implementation on the ASR 1000 Series Aggregation Services Router as a *three-parameter scheduler*.

In an existing IOS classic implementation, we provide a simpler *two-parameter scheduler*. Instead of distinct entries for Min and Ex, each schedule entry has only a single weight. Whether you used the **bandwidth** or

bandwidth remaining command you were configuring the same single weight. To grasp the difference, let's look at an example focusing on the **bandwidth** command.

The policy-map for this example will look as follows:

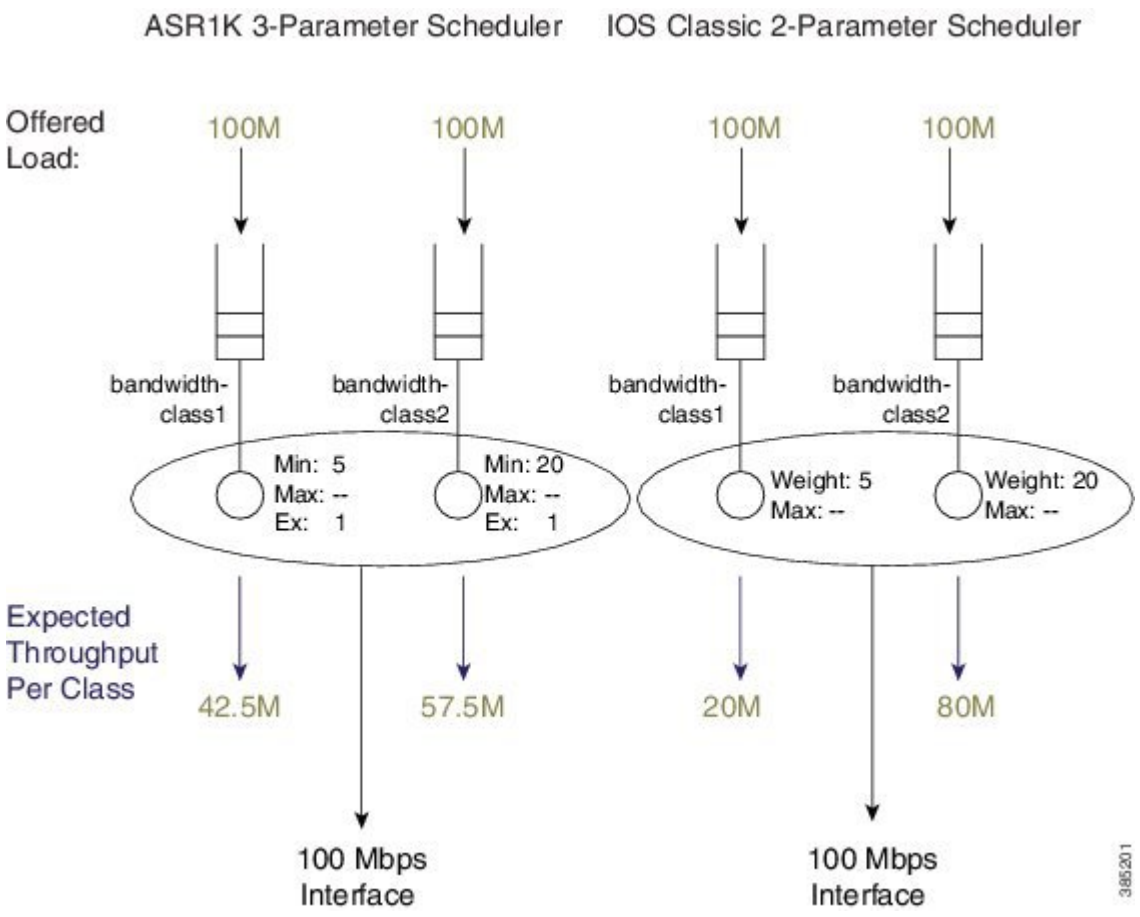
```
policy-map bandwidth-example
  class bandwidth-class1
    bandwidth percent 5
  class bandwidth-class2
    bandwidth percent 20
```

Now consider the policy-map attached to a 100 Mbps interface and offer 100 Mbps to each queue. The following figure shows how the schedule configuration and expected throughput would appear both on an ASR 1000 Series Aggregation Services Router (on the left) and on a router running an IOS Classic image (e.g., a Cisco 7200; on the right).

25 Mbps is assigned to the ASR 1000 Series Aggregation Services Router three-parameter scheduler and we use it to honor Min guarantees, which leaves 75 Mbps of excess bandwidth. This excess is shared equally between the two queues based on the default excess-weight of '1' that each queue will receive.

If we apply the same configuration on a two-parameter scheduler, the configured bandwidth values will dictate a single-weight parameter in the schedule entry. The concept "Min scheduling versus excess bandwidth sharing" does not apply here. Instead, for each entry, all bandwidth sharing hinges on the single weight.

Figure 16: Same Configuration running on ASR 1000 with IOS XE and Router running IOS Classic



Looking at this example you see that the same configuration on an ASR 1000 Series Aggregation Services Router running IOS XE can yield significantly different behavior from a router running an IOS classic image.

Note To achieve identical behavior to an router running IOS Classic, you can use a configuration using only excess bandwidth sharing. Changing **bandwidth percent** statements in an IOS Classic configuration to **bandwidth remaining percent** statements in IOS XE is an easy way to migrate existing configurations.

Schedule Burstiness

Possible sources of burstiness in scheduling include: packet batching and the scheduler's representation of time.

Packet Batching

This source is intentional and should not cause concern. As implemented in hardware, we cap the number of decisions a schedule can make per second. If you were to send all small packets, say 64-byte frames, a schedule may struggle to maintain if it is making decisions, packet by packet, for a fast interface like 10 Gbps. To ease this burden, the hardware will batch small packets (up to about 512 bytes from the same queue) and let the scheduler treat them as a single decision.

So, if a single packet of 512 bytes were at the head of the queue we would send that to the schedule as a single packet. On the contrary, if five 64-byte packets were at the head of the queue we would batch the packets as a single packet from the scheduler's perspective. That is, we would pull all five packets from the queue simultaneously and forward them back to back on the wire as a single burst. As the size of a single MTU greatly exceeds that of a burst, the later negligibly impacts downstream buffering or jitter for other queues.

Scheduler's Representation of Time

The second potential source of burstiness arises from how a schedule in hardware tracks time. If you mix very small rates (say 100K and less) and very large rates (say 100M and higher) in the same policy-map you may experience unexpected burstiness in scheduling of traffic from the queue configured with the high rate.

When you use *real-time scheduling* (using either the **bandwidth** or **shape** command) you are specifying rates in bits per second. This means that each schedule entry must have a concept of real time and must be monitoring service rate vs that real time. The representation of time must be uniform across all entries in a given scheduler.

Consider an 8-Kbps shaper (8000 bits/sec = 1000 bytes/sec).

Sending 64-byte packets would be (equivalent to) sending one packet every (64-byte packet * 64/1000 =) 64 mS.

Sending 1500-byte packets would be (equivalent to) sending one packet every 1500 byte * 1500/1000 =) 1.5 sec.

We want to represent anything ranging from 64 mS to many seconds. To do this we count time and establish that every counter increment represents 10.5 mS of real time.

Now consider a 10-Gbps shaper. In 10.5 mS we would expect to send 8,750 1500-byte packets:

10,000,000,000 b/sec * .0105 sec = 105,000,000 bits, which equals 105,000,000/8/1500, or 8750 1500-byte packets

This is a huge data burst. If we were counting time in 10.5 mS increments, whenever the clock (advances) we would need to send that burst. In contrast, if .65 mS represented *real time*, we would expect to send 542 1500-byte packets (a far more manageable situation).

The representation of time is driven by the lowest rate configurable within a policy-map. The following table shows the granularity of time chosen vs. rate that is configured in a policy-map. (The details are accurate for ESP-20 but only similar for all variants of ASR1K hardware.)

Reading the table you observe that if all rates in your policy-map are 116K or greater, then any burst introduced by this representation of time would last less than a millisecond and therefore insignificant. If you configure shape or bandwidth rates less than 116K on a fast interface you may want to ensure there are no unintended consequences. (e.g., if all rates in your policy-map range from 29K - 57K, then any burst introduced by this representation of time would be 2.6 mS in duration!) Such consequences could include downstream devices dropping if they have insufficient buffering to receive such a burst, WRED dropping packets, or downstream policers dropping packets due to exceeding their burst allowance.

Minimum Guaranteed Service Rate for a Queue

For any queue you can calculate a minimum guaranteed service rate (the service that a queue will receive if all other queues are congested). In some prior examples we have shown an offered rate of 100% to each queue - the expected throughput in those examples is the minimum guaranteed service rate. Knowing this rate might inform you on how your applications will behave under severe congestion. That is, when the network is systemically overloaded, can you expect your application to run?

One particularly useful number you can calculate from the guaranteed rate is the delay a packet would experience if the egress queue were full. For example, let's consider an oversubscribed video queue, whose policy-map looked as follows:

```
policy-map min-service-rate-example
  class priority-class
    priority
    police cir 1m 1250
  class video
    bandwidth 1000
  class mission-critical
    bandwidth 2000
```

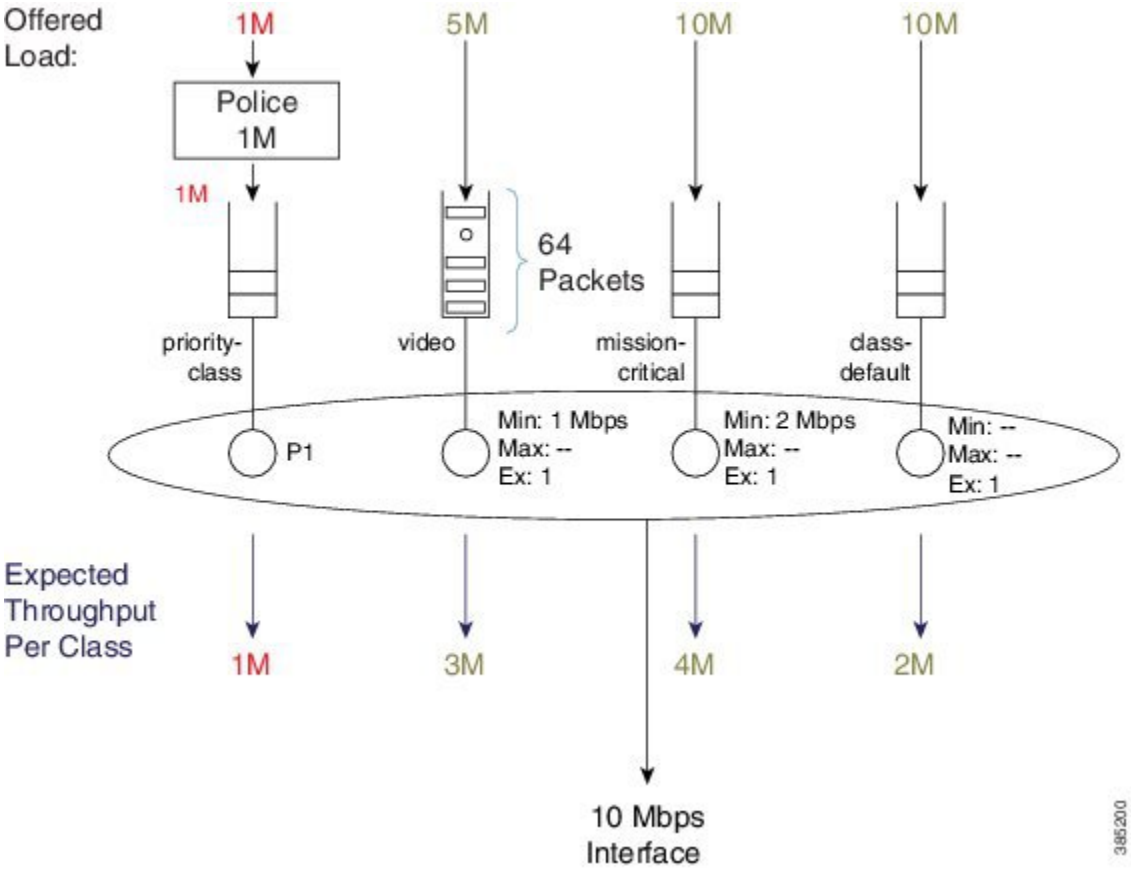
In this example we attach the policy-map to a 10 Mbps Ethernet interface and the offered load to each class as shown in the following analysis. (We will abide by the guidelines in Schedule Operation, on page 34.)

The video class is serviced at its minimum guaranteed service rate of 3 Mbps (1 Mbps (Min configured with the **bandwidth** command) + 2 Mbps (its always guaranteed share of excess bandwidth). Looking more closely at the calculations involved:

10 Mbps - 1 Mbps(for P1) - 3 Mbps(Min guaranteed for video and mission-critical) = 6 Mbps

6 Mbps excess bandwidth(after accounting for P1 and Min guarantees)/3 equal shares for all queues = 2 Mbps

Figure 17: Minimum Service Rate and "Experience" of Latency



With the minimum guaranteed service rate, you can now calculate "the experience" of latency packets arriving at the video queue. Using the default queue-limit of 64 packets (under over-subscription, we would expect the queue to fill and contain 64 packets) a new packet would be either dropped or placed at the tail of the queue (if the packet arrived just when another was pulled from the video queue).

Given this queue for video traffic, we could expect the average packet size to be about 1400 bytes (roughly the size of an MPEG I-frame), generating 716,800 bits as the amount of data buffered and awaiting transmission:

64 packets * 1400 bytes/packet * 8 bits/byte = 716,800 bits

Given a minimum rate of 3 Mbps, we would require 239 mS to drain this queue:

716,800 bits / 3 Mbps = 0.239 Seconds (239 mS)

As you can see, a minimum guaranteed service rate can help you envisage (and so predict) the behavior of your applications under congested conditions.

Pak Priority

Pak priority designates a scheme to protect some critically important control packets (interface keepalives, BFD packets, some routing protocol hellos etc.) that are vital for network stability. In this section, we will describe these packets and outline how they are scheduled.

Note The name, pak_priority, is unfortunate and might cause confusion because the control packets are not (actually) queued in a priority queue.

With pak_priority, we attempt to ensure guaranteed delivery and not guaranteed low latency) of the control packets. They are marked with an internal pak_priority flag, when first generated in the control plane. This flag does not propagate outside the router and is only used to ensure we give special treatment to the packet as we send it towards the egress interface.

Observe that we set the DSCP of IP encapsulated control packets to CS6 protecting them at other devices in the network where they must traverse. On the router that generates the control packet, the pak_priority designation dictates further protection beyond what you might configure for CS6 packets.

The following table lists the packets and protocols we mark with the internal pak_priority flag.

Packets and Protocols Marked with the pak_priority Flag

Table 2: Control Packets Marked with pak_priority Flag

Level Marked	Packets and Protocols
Layers 1 and 2	
	ATM Address Resolution Protocol Negative Acknowledgment (ARP NAK)
	ATM ARP requests
	ATM host ping operations, administration and management cell(OA&M)
	ATM Interim Local Management Interface (ILMI)
	ATM OA&M
	ATM ARP reply
	Cisco Discovery Protocol
	Dynamic Trunking Protocol (DTP)
	Ethernet loopback packet
	Frame Relay End2End Keepalive
	Frame Relay inverse ARP
	Frame Relay Link Access Procedure (LAPF)
	Frame Relay Local Management Interface (LMI)
	Hot standby Connection-to-Connection Control packets (HCCP)

Level Marked	Packets and Protocols
	High-Level Data Link Control (HDLC) keep-alives
	Link Aggregation Control Protocol (LACP) (802.3ad)
	Port Aggregation Protocol (PAgP)
	PPP keep-alives
	Link Control Protocol (LCP) Messages
	PPP LZS-DCP
	Serial Line Address Resolution Protocol (SLARP)
	Some Multilink Point-to-Point Protocol (MLPP) control packets (LCP)
IPv4 Layer 3	
	Protocol Independent Multicast (PIM) hellos
	Interior Gateway Routing Protocol (IGRP) hellos
	OSPF hellos
	EIGRP hellos
	Intermediate System-to-Intermediate System (IS-IS) hellos, complete sequence number PDU (CSNP), PSNP, and label switched paths (LSPs)
	ESIS hellos
	Triggered Routing Information Protocol (RIP) Ack
	TDP and LDP hellos
	Resource Reservation Protocol (RSVP)
	Some L2TP control packets
	Some L2F control packets
	GRE IP Keepalive
	IGRP CLNS
	Bidirectional Forwarding Protocol (BFD)

Levels of Protection for pak_priority Packets

First level

A policer or WRED will not drop packets with this designation.

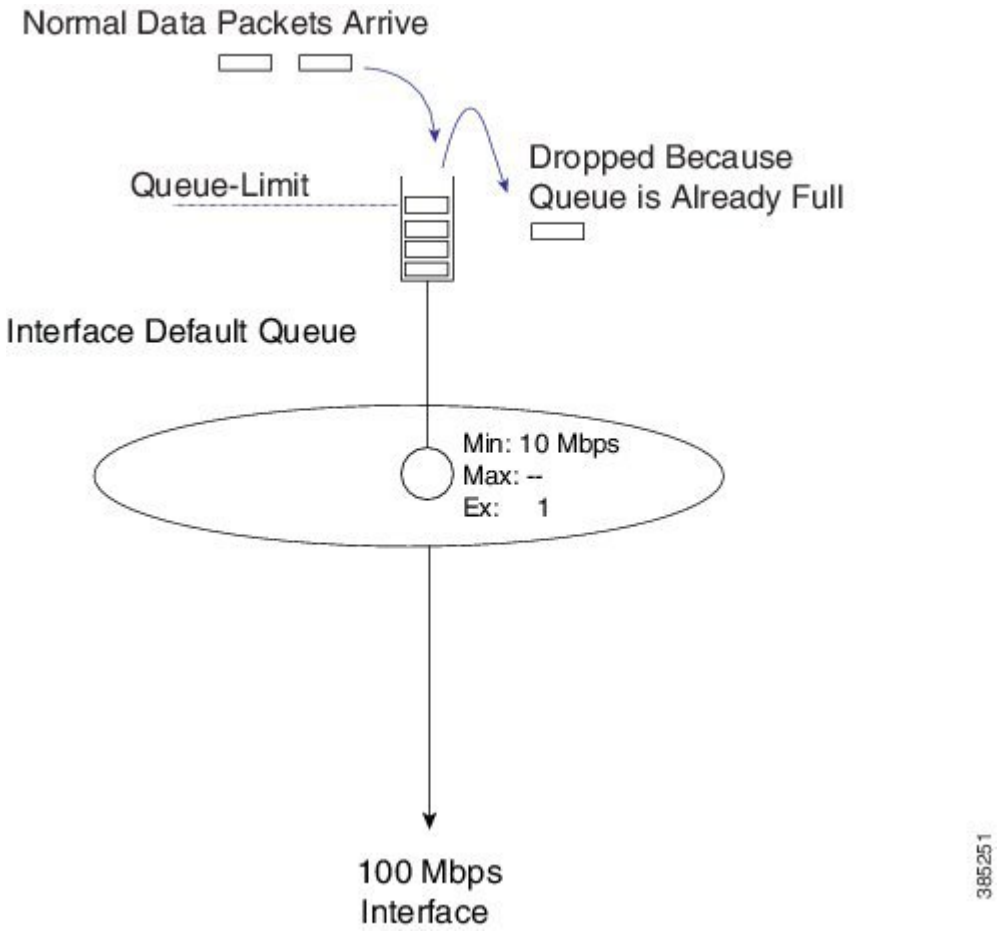
Second level

We will enqueue pak_priority packets even if an egress queue is already full. To grasp this let's first look at a physical interface that has no QoS configured, where we still need a queue and (therefore) a schedule to pull packets from that queue.

Without QoS configured on the interface, we have a single first-in first-out (FIFO) queue (referred to as the *interface default queue*). (Do not confuse with a *class-default queue*).

In the diagram below, normal data packets arrive when the queue is full. Notice that the schedule entry has the typical Min and default Ex values (10% of the interface rate and 1). Because only one queue exists, these values have no effect; without competition, one queue will receive all the available bandwidth.

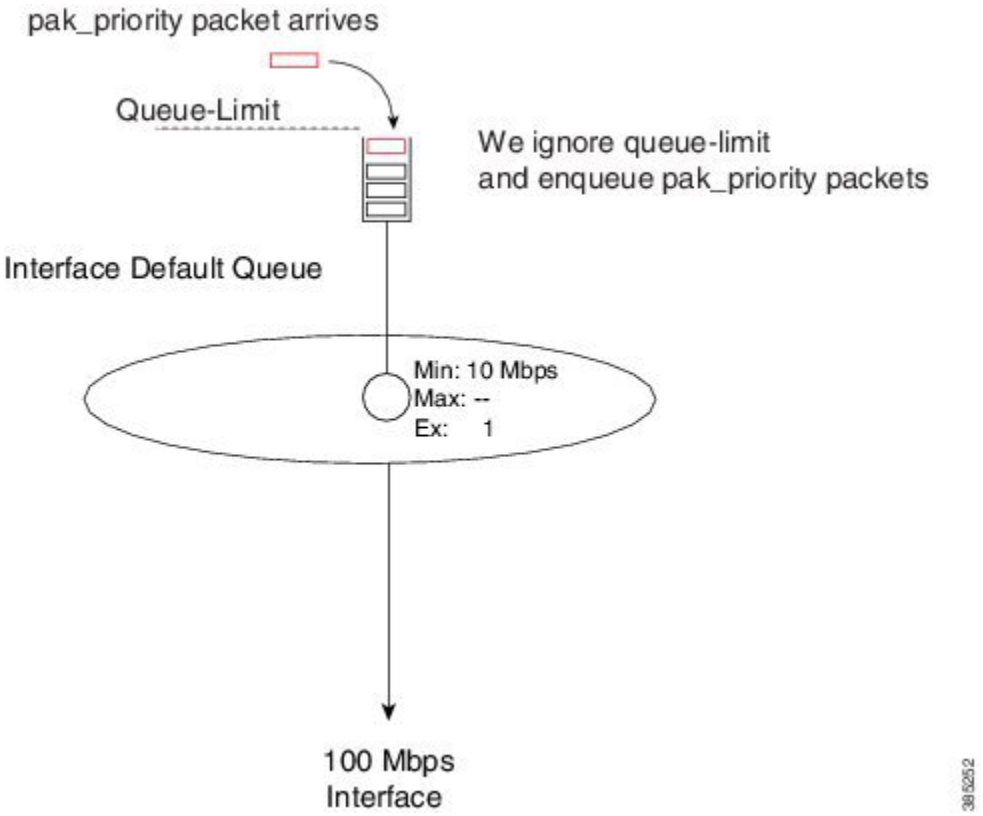
Figure 18: Normal Packet Arrives when Queue is Full



As with every queue, a queue-limit determines how much data we can buffer before the queue is considered full.

If a pak_priority packet arrives while the queue is full, we ignore the queue-limit and enqueue it; the packet must wait until all leading packets are sent. On most ASR 1000 Series Aggregation Services Router platforms, the default queue-limit is 50 mS. So, we may delay the pak_priority packet for 50 mS but we do guarantee its delivery.

Figure 19: pak_priority Packet Arrives when Queue is Full, and we Ignore queue-limit

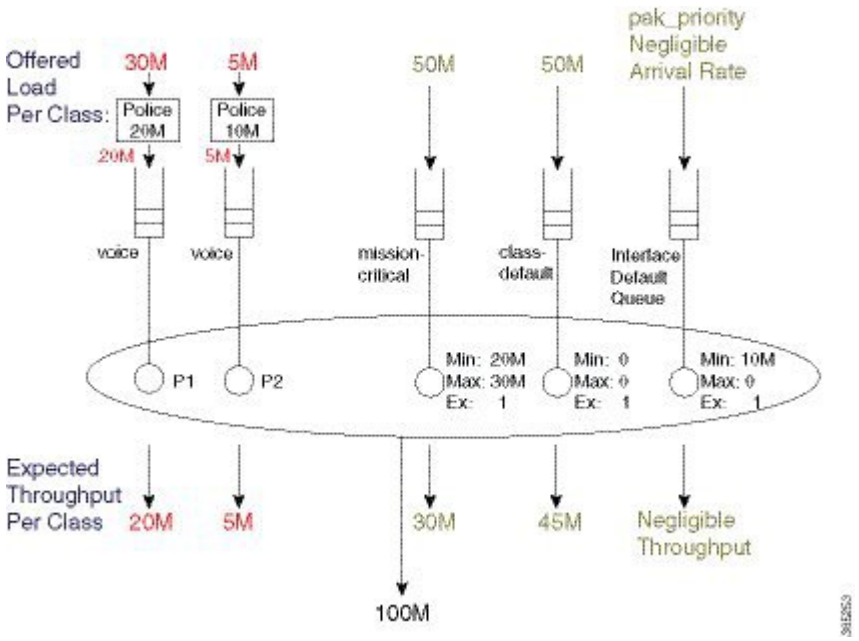


The discussion (of pak priority) changes slightly when you configure QoS on an interface. To illustrate, let's reconsider one of the very first examples in this chapter (see Schedule Operation: With a Shaper, on page 36).

```
policy-map scheduling-example
  class voice
    priority level 1
    police 20m
  class video
    priority level 2
    police 10m
  class mission-critical
    bandwidth 20000
    shape average 30m
```

We didn't show it previously and as you will now observe below, when you configure a policy we do not remove the interface default queue.

Figure 20: Repurposing Interface Default Queue as a Dedicated pak_priority Queue



Pak_priority packets are still enqueued in the interface default queue. Essentially, the queue has been repurposed as a dedicated pak_priority queue.

Looking at the diagram you can now see the importance of the Min value (set at 10% of linerate). This value ensures that other queues (with a Min service configured) cannot deprive this queue of service.

However, although we configure a Min of 10% of linerate we will never consume this bandwidth. We only mark a very small number of critical packets as pak_priority so the rate at which they arrive is negligible. We overpromise on the Min with the understanding that it will not impact behavior of other queues. Were we to mark too many packets "pak_priority," this scheme would not work.

With routing protocols we mark Hello packets but not routing updates with pak_priority. So, you should create a bandwidth queue for CS6 packets.

The Hello packets will pass through the interface default queue and the routing updates will use the newly-created bandwidth queue.

An exception is BGP, where we do not mark Hello packets as pak_priority. Why? BGP Hello packets and updates share the same TCP stream. Providing special treatment would cause TCP packets to arrive out of sequence. This offers no benefit as you could not consume them.

Classifying a pak_priority packet (by matching DSCP or any other field) and attempting to move it to a bandwidth queue will not happen - the packet will still be enqueued in the interface default queue. However, you can classify the packet and move it to a designated priority queue.

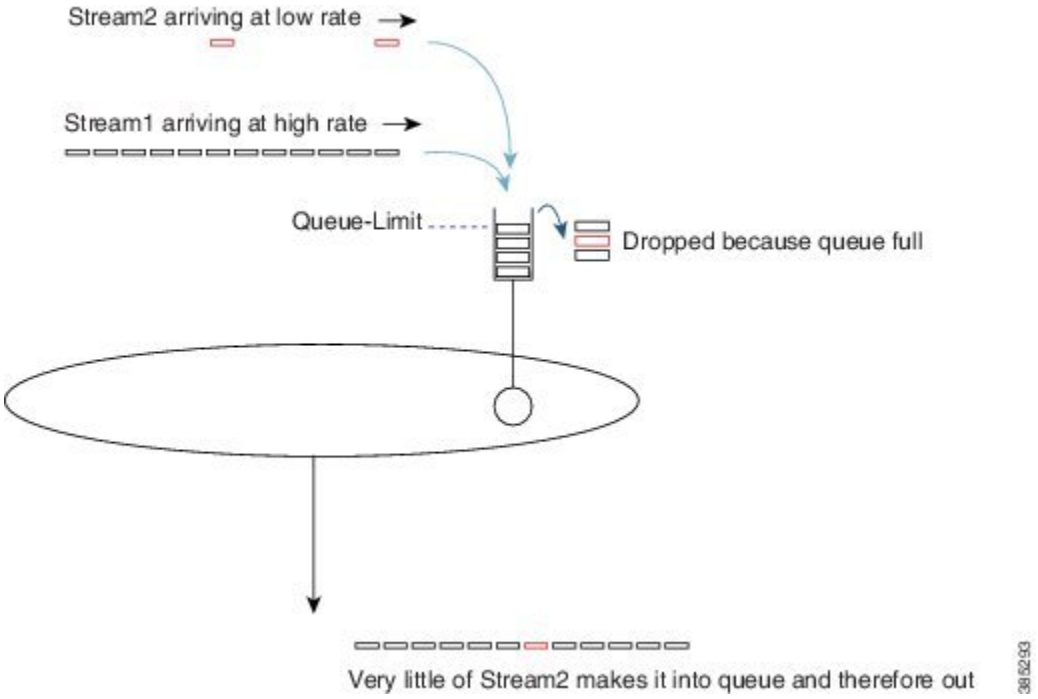
Flow-Based Fair Queuing

Flow-based fair queuing enables us to provide some fairness between flows that belong to the same class in a policy-map. It provides some protection for low speed (well behaved) streams of traffic competing with high-rate streams.

If an individual class is oversubscribed (the offered rate exceeds the service rate), one high-rate flow can starve low-rate flow(s) of service. To understand this you need to consider multiple streams targeting the same physical queue. After it fills, additional packets are dropped. Whenever the scheduler sends a packet from the queue, a single space will open at the queue's tail. We successfully enqueue whichever packet arrives next. If you look at the following example, you notice that a packet from Stream1 (the high rate stream) is more likely to arrive next. Fairness between treatment of the streams is absent! Whatever exits the interface is purely driven by what packet manages to make it into the queue.

Not only does the high rate stream obtain an unfair share of the class bandwidth, it also impacts latency for low rate streams. As successfully-enqueued packets are always at the tail of a full queue, they must wait for all other packets to drain before they can be transmitted.

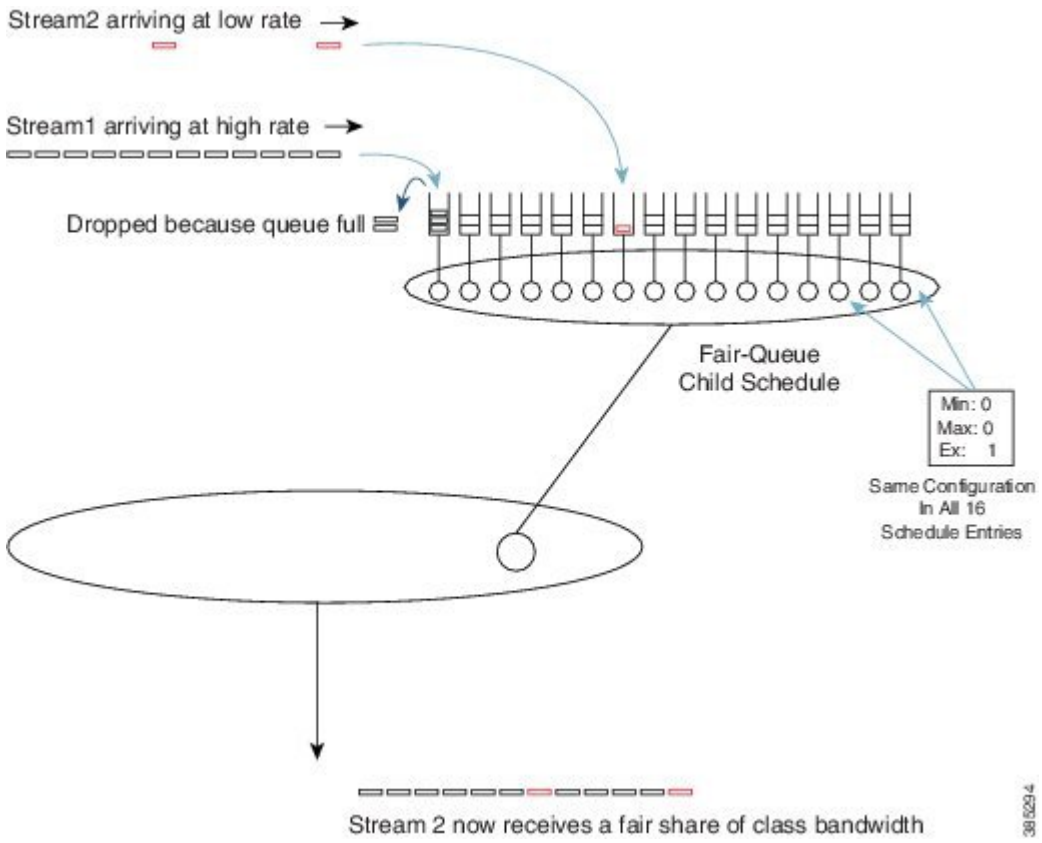
Figure 21: Latency without Flow-Based Fair Queuing



Flow-based fair queuing can alleviate this issue. When you issue the **fair-queue** command you direct the router to create 16 queues for that single class, which represents a simple form of *hierarchical scheduling*. .

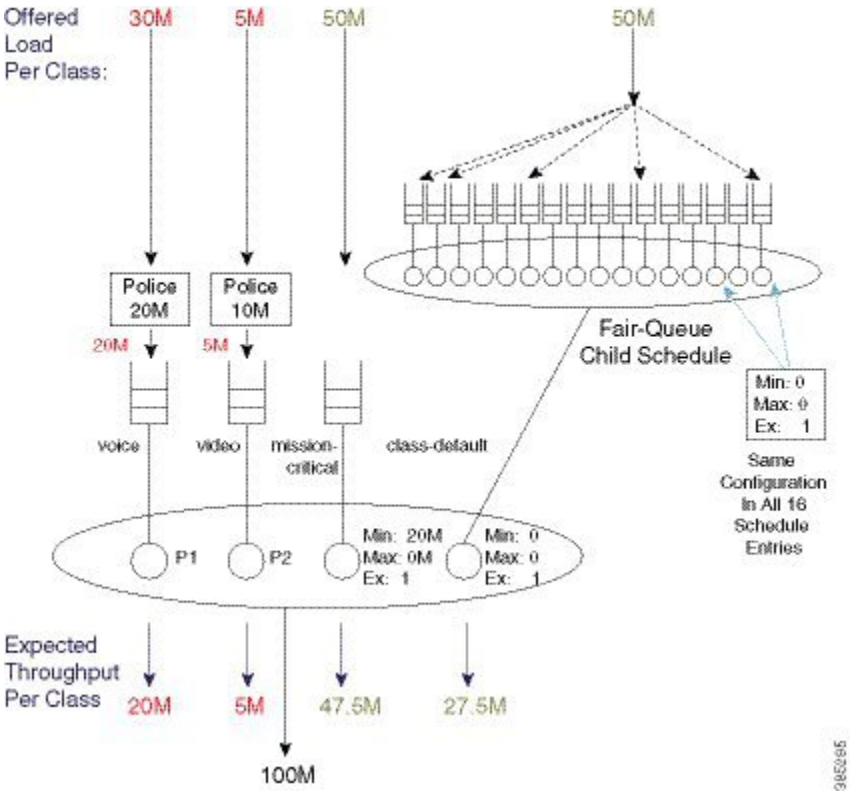
Consider the fair-queue child schedule a *pre-sorter*, providing sorting and fairness between multiple streams targeting the same class.

Figure 22: Pre-sorting and Fairness provided by Flow-Based Fair Queuing




```
policy-map fair-queue-example
  class voice
    priority level 1
    police 20m
  class video
    priority level 2
    police 10m
  class mission-critical
    bandwidth 20000
  class class-default
    fair-queue
```

Figure 23: Packet Flow with Flow-Based Fair Queuing



Verification

Use the **show policy-map interface** *interface* command to verify operation of scheduling. This command will show long term trends and a complete view of the policy configured.

The data plane sends statistics to the control plane every 10 seconds and control plane refreshs its own statistics every 10 seconds. This means that the values in output of **show policy-map interface** command updates every 10 seconds. Some counters that represent instantaneous state, such as current queue depth, may not be overly useful. It is possible to look directly at hardware counters if you really want true instantaneous state.

The following configuration is an example of **show policy-map interface***interface* command.

```
policy-map show_policy-example
 class voice
  priority level 1
  police cir percent 10 bc 5 ms
 class video
   priority level 2
  police cir percent 20 bc 10 ms
 class critical-data
  bandwidth percent 50
```

This policy has four classes, the three that are explicitly configured and the implicit class-default.

The output from the **show policy-map interface** command mirrors the configured policy. It has a section for each configured class. Within each class the output is consistently organized with a classification section and a section for each configured action.

Note that queuing information for priority classes is shown separately to other features (policers) in that class. This is since multiple priority classes may map into the same queue.

The following is an example of the output from the **show policy-map interface** command. Although it is one continuous block of output we break it into sections to highlight the structure of the output.

`Device#show policy-map interface g1/0/4`
` GigabitEthernet1/0/4`
 	
`  Service-policy output: show_policy-example`
 	
`queue stats for all priority classes:`
`  Queueing`
`  priority level 1`
`  queue limit 512 packets`
`  (queue depth/total drops/no-buffer drops) 0/0/0`
`  (pkts output/bytes output) 39012/58518000`
	This section shows queue information for the priority level 1 queue

`  queue stats for all priority classes:`
`  Queueing`
`  priority level 2`
`  queue limit 512 packets`
`  (queue depth/total drops/no-buffer drops) 0/0/0`
`  (pkts output/bytes output) 61122/91683000`
	This section shows queue information for the priority level 2 queue

`Class-map: voice (match-all)`
`  39012 packets, 58518000 bytes`
`  5 minute offered rate 672000 bps, drop rate 0000 bps`
`  Match:  dscp ef (46)`
	This section shows statistics for the class named voice.
	First shown is the classification statistics.

`Priority: Strict, b/w exceed drops: 0`
`  Priority Level: 1`
 	
`  police:`
`  cir 10 %, bc 5`
`  cir 100000000 bps, bc 62500 bytes`
`  conformed 39012 packets, 58518000 bytes; actions:`
`  transmit`
`  exceeded 0 packets, 0 bytes; actions:`
 	
`  drop`
`  conformed 672000 bps, exceeded 0000 bps`
	Priority level indicates the priority queue above that will be used by this class.
	The statistics for the policer used for queue admission control are also shown here.

`Class-map: video (match-all)`
`1376985 packets, 2065477500 bytes`
`5 minute offered rate 9171000 bps, drop rate 0000 bps`
`Match: dscp af41 (34)`
	This is start of section for class named video.
	Again classification statistics and criteria are shown first

`police:`
`cir 20 %, bc 10`
`cir 200000000 bps, bc 250000 bytes`
`conformed 1381399 packets, 2072098500 bytes; actions:`
`transmit`
`exceeded 0 packets, 0 bytes; actions:`
`drop`
`conformed 9288000 bps, exceeded 0000 bps`
`Priority: Strict, b/w exceed drops: 0`
`Priority Level: 2`
	This section shows actions configured in the class named video.
	The statistics for the queue admission control policer.
	Priority Level indicates packets from this class will be enqueued in the priority level 2 queue that is shown above.

`Class-map: critical-data (match-all)`
`45310 packets, 67965000 bytes`
`5 minute offered rate 719000 bps, drop rate 0000 bps`
`Match: dscp af11 (10)`
	This is start of section for class named critical-data.
	As always the classification statistics and criteria are shown first.

`Queueing`
`queue limit 2083 packets`
`(queue depth/total drops/no-buffer drops) 0/0/0`
`(pkts output/bytes output) 45310/67965000`
`bandwidth 50% (500000 kbps)`
	Since this class has the bandwidth action a queue is created for the class.
	This section shows the queue related configuration and statistics.

`Class-map: class-default (match-any)`
`51513 packets, 77222561 bytes`
`5 minute offered rate 194000 bps, drop rate 0000 bps`
`Match: any`
	This is the start of the section for class-default, the implicit class that exists in every policy.
	As always we first show the statistics for packets deemed to belong to this class.

`queue limit 4166 packets`
`(queue depth/total drops/no-buffer drops) 0/0/0`
`(pkts output/bytes output) 1371790/2057638061`
	This section shows the queue information for class-default.

As mentioned above the **show policy-map interface** command receives an update from the data plane every 10 seconds.

It is also possible to look directly at the data plane for a real time view of system behavior. The command will also allow you to verify the data plane is programmed as expected.

Counters for long term events such as packets enqueued or packets dropped will be cleared each time that information is pushed to the control plane, every 10 seconds.

Perhaps the most useful counter in the data plane show output is the instantaneous queue depth. As this is read each time you issue the command you can get realtime visibility into whether a queue is congested, is it sustained congestion or bursty behavior etc.

The **show platform hardware qfp active feature qos interface** *interface* is the command to view QoS configuration and statistics in the hardware data plane.

The following show example output from the command corresponding to the configuration and **show policy-map interface** example above.

You can see the output of the command again reflects the structure of the policy-map with a section for each class configured.

`Device#show platform hardware qfp active feature qos interface gig1/0/4`
`Interface: GigabitEthernet1/0/4, QFP interface: 11`
`  Direction: Output`
`    Hierarchy level: 0`
`    Policy name: show_policy-example`	

`      Class name: voice, Policy name: show_policy-example`
`        Police:`
`            cir:  100096000 bps,  bc: 63488 bytes`
`            pir: 0 bps,  be: 0 bytes`
`            rate mode: Single Rate Mode`
`            conformed: 0 packets, 0 bytes; actions:`
`              transmit`
`            exceeded: 0 packets, 0 bytes; actions:`
`              drop`
`            violated: 0 packets, 0 bytes; actions:`
`              drop`
`            color aware: No`
`            green_qos_group: 0, yellow_qos_group: 0`
`            overhead accounting: disabled`
`            overhead value: 0, overhead atm: No`
	Start of section for class named voice.
	Policer configuration and statistics for current 10 second interval

`Queue: QID: 175 (0xaf)`
`bandwidth (cfg) : 0`
`, bandwidth (hw) : 0`
`shape (cfg) : 0`
`, shape (hw) : 0`
`prio level (cfg) : 1`
`, prio level (hw) : 0`
`limit (pkts ) : 512`
`drop policy: tail-drop`
`Statistics:`
`depth (pkts ) : 0`
`tail drops (bytes): 0`
`, (packets) : 0`
`total enqs (bytes): 0`
`, (packets) : 0`
`licensed throughput oversubscription drops:`
`(bytes): 0`
`, (packets) : 0`
`Schedule: (SID:0x258)`
`Schedule FCID : 16`
`bandwidth (cfg) : 1050 Mbps`
`, bandwidth (hw) : 1050.01 Mbps`
`shape (cfg) : 1050 Mbps`
`, shape (hw) : 1050.01 Mbps`
	Queue information for class voice
	This depth is instantaneous queue depth – can be very useful

`Class name: class-default, Policy name: show_policy-example`
`Queue: QID: 176 (0xb0)`
`bandwidth (cfg) : 0`
`, bandwidth (hw) : 0`
`shape (cfg) : 0`
`, shape (hw) : 0`
`prio level (cfg) : 0`
`, prio level (hw) : n/a`
`limit (pkts ) : 4166`
`drop policy: tail-drop`
`Statistics:`
`depth (pkts ) : 0`
`tail drops (bytes): 0`
`, (packets) : 0`
`total enqs (bytes): 3420000`
`, (packets) : 2280`
`licensed throughput oversubscription drops:`
`(bytes): 0`
`, (packets) : 0`
	Start of section for class-default.
	Queue information for this class.
	Instantaneous depth and statistics for current 10 second interval

`Class name: video, Policy name: show_policy-example`
`Police:`
`cir: 200064000 bps, bc: 253952 bytes`
`pir: 0 bps, be: 0 bytes`
`rate mode: Single Rate Mode`
`conformed: 0 packets, 0 bytes; actions:`
`transmit`
`exceeded: 0 packets, 0 bytes; actions:`
`drop`
`violated: 0 packets, 0 bytes; actions:`
`drop`
`color aware: No`
`green_qos_group: 0, yellow_qos_group: 0`
`overhead accounting: disabled`
`overhead value: 0, overhead atm: No`
	Start of section for class named video.
	Admission control policer configuration and statistics are shown first.

`Queue: QID: 178 (0xb2)`
`bandwidth (cfg) : 0`
`, bandwidth (hw) : 0`
`shape (cfg) : 0`
`, shape (hw) : 0`
`prio level (cfg) : 2`
`, prio level (hw) : 1280`
`limit (pkts ) : 512`
`drop policy: tail-drop`
`Statistics:`
`depth (pkts ) : 0`
`tail drops (bytes): 0`
`, (packets) : 0`
`total enqs (bytes): 0`
`, (packets) : 0`
`licensed throughput oversubscription drops:`
`(bytes): 0`
`, (packets) : 0`
`Schedule: (SID:0x258)`
`Schedule FCID : 16`
`bandwidth (cfg) : 1050 Mbps`
`, bandwidth (hw) : 1050.01 Mbps`
`shape (cfg) : 1050 Mbps`
`, shape (hw) : 1050.01 Mbps`
	Queue information for class video
	Instantaneous queue depth and statistics for current 10 second interval

`Class name: critical-data, Policy name: show_policy-example`
`  Queue: QID: 177 (0xb1)`
`  bandwidth (cfg)  : 500000000`
`  , bandwidth (hw)  : 500000000`
`  shape (cfg)  : 0`
`  , shape (hw)  : 0`
`  prio level (cfg)  : 0`
`  , prio level (hw)  : n/a`
`  limit (pkts )  : 2083`
`  drop policy: tail-drop`
`  Statistics:`
`  depth (pkts )  : 0`
`  tail drops (bytes): 0`
`  , (packets)  : 0`
`  total enqs (bytes): 0`
`  , (packets)  : 0`
`  licensed throughput oversubscription drops:`
`  (bytes): 0`
`  , (packets)  : 0`
	Start of section for class named critical-data.
	Instantaneous queue depth and statistics for current 10 second interval.

Command Reference

Account

Account is not an independent command but rather an extension to scheduling commands that allows a user to specify overhead accounting for that command. Account is presented here to avoid replication in each of the scheduling commands.

Syntax description:

To configure a user defined number of bytes to be added to or subtracted from the scheduling length:

[**no**] **shape** | **bandwidth** *rate* **account user-defined** *value* [**atm**]

To specify encapsulation of a downstream device and automatically calculate the overhead accounting adjustment:

[**no**] **shape** | **bandwidth** *rate* **account dot1q** | **qing** *encapsulation*

Command Default:

By default Layer 3 Datagram and Layer 2 headers are included in scheduling calculations.

Usage Guidelines:

If the account option is used in one class containing scheduling actions in a policy-map, the account command with same values must be used in all classes containing scheduling actions. Similarly in a hierarchical policy-map the same account options must be configured in each level of the policy.

Bandwidth

The bandwidth command is used to guarantee a minimum service rate to a class.

Syntax description:

To configure in Kbps:

[**no**] **bandwidth** *rate* [**account** *account options*]

To configure as a percentage of visible bandwidth:

[**no**] **bandwidth percent** *value* [**account** *account options*]

Command Default:

By default there is no minimum bandwidth value configured in the schedule entry for a queue. Note that the default excess weight does guarantee some minimum service.

Usage Guidelines:

The **bandwidth** command may be useful if you have an application for which you know the minimum bandwidth requirements.

Bandwidth rates can be configured in 8Kbps increments and the ASR1K has been tested to achieve accuracy within 1% of those rates.

The **bandwidth** command is only supported in leaf schedules (class layer schedules). If you wish to apportion bandwidth in a parent policy you may use the **bandwidth remaining** command.

If you wish to replicate scheduling behavior of an IOS Classic platform (2 parameter scheduler) you may want to replace all **bandwidth percent** *value* commands in your configuration with **bandwidth remaining percent** *value* command.

Bandwidth remaining

The **bandwidth remaining** command is used to apportion excess bandwidth between classes. It may be configured as a simple weight or as a percentage of available bandwidth.

Syntax Description:

To configure as a simple weight:

[**no**] **bandwidth remaining ratio** *value* [**account** *account options*]

To configure as a percentage:

[**no**] **bandwidth remaining percent** *value* [**account** *account options*]

Command Default:

By default every bandwidth schedule entry, whether in leaf schedule or a parent schedule, is configured with an excess weight of 1. This is equivalent to **bandwidth remaining ratio** *1* being configured in that class.

Usage Guidelines:

Configuring bandwidth remaining as a weight supports values of 1 to 1000. This can allow more granular excess sharing than using the percent option.

Configuring **bandwidth remaining percent** *value* yields behavior similar to IOS classic which used a 2 parameter scheduler.

With a shape on parent / queue on child policy (parent has only class default) **bandwidth remaining ratio** *value* should be used to apportion bandwidth between logical interfaces where parent polices are attached

Fair-Queue

The **fair-queue** command is used to configure flow based fair-queuing in a class configured as a bandwidth queue.

Syntax Description:

fair-queue

Command Default:

By default a single fifo queue is configured for each bandwidth class.

Usage Guidelines:

Flow based fair-queuing is used to ensure a single greedy flow can't consume all the bandwidth allocated to a class.

All packets from any given flow are hashed into the same flow queue.

Flow-queuing should not be configured in a policy attached to a tunnel interface. Since all packets have the same outer header all packets are hashed to the same flow queue thus rendering the feature ineffective.

Priority

The **priority** command is used to give low latency and low jitter treatment to a class of traffic.

Syntax Description:

To configure an absolute priority queue (note should be used with explicit policer)

[**no**] **priority**

To configure an absolute priority queue with multi-level priority queuing (note this should be used with an explicit policer)

[**no**] **priority level** *1* | *2*

To configure a priority queue with conditional policer

[**no**] **priority** *rate in kbps* [**burst in bytes**]

or

[**no**] **priority percent** *rate* [**burst in bytes**]

To configure multilevel priority queueing with conditional policer

[**no**] **priority level** *1* | *2rate in kpbs* [**burst in bytes**]

or

[**no**] **priority level** *1* | *2* **percent***rate* [**burst in bytes**]

Command Default:

By default queues are not configured with priority treatment.

Usage Guidelines:

Priority queues should be used with some form of queue admission control (explicit policer or conditional policer) to avoid chance of starving other classes of service.

The policer conforming burst should be configured to an appropriate value for the application in the queue. The following is a configuration example

```
policy-map always_on_policer_burst_example
    class voice
      priority
      police cir 2000000 1250
```

It is not necessary to configure priority in the parent of a hierarchical policy as priority propagation will ensure packets marked as priority by a leaf schedule will receive priority treatment throughout the scheduling hierarchy.

Shape

Use the **shape** command to configure the maximum rate at which a queue may be serviced. Configuring a shaper does not guarantee throughput to a class, it simply puts an upper bound on the rate at which that class may be serviced.

Syntax Description:

[**no**] **shape average** *rate* [**unit**] [**confirming burst**] [**excess burst**] [**account** *options*]

or

[**no**] **shape average percent** *rate* [**confirming burst**] [**excess burst**] [**account** *options*]

Command Default:

By default there is no maximum rate configured in the schedule entry for a bandwidth queue.

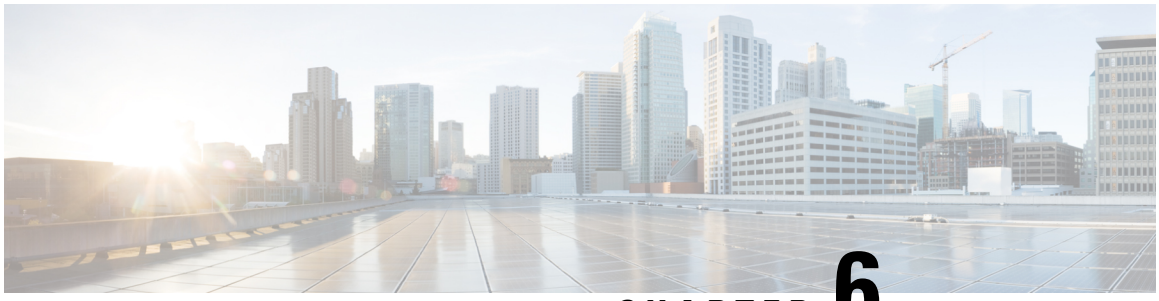
Usage Guidelines:

The **shape** command is most commonly used in a parent policy to limit the rate at which traffic is sent to a remote site.

When used in a parent policy the shape rate will be enforced for all traffic - priority and bandwidth.

The **shape** command has options for conforming and excess burst sizes but these values have no effect on XE platforms. Hardware scheduling on the ASR1K platform obviates the need to optimize burst parameters.

The **shape** command enforces a maximum rate at which a class may be serviced but does not in itself guarantee any throughput to that class. The **bandwidth remaining** command may be used along with the shape command to guarantee throughput.



CHAPTER 6

QoS Hierachical Scheduling

In this chapter we will see how commands and their semantics as covered in the scheduling chapter can be combined in different ways to achieve more complex outcomes.

Two distinct approaches are available to configure complex scheduling hierarchies: hierarchical policy-maps and policy-maps attached to logical interfaces. Here, we illustrate how either can achieve the same outcome and delineate the relative benefits of each approach.

About Hierarchical Schedules

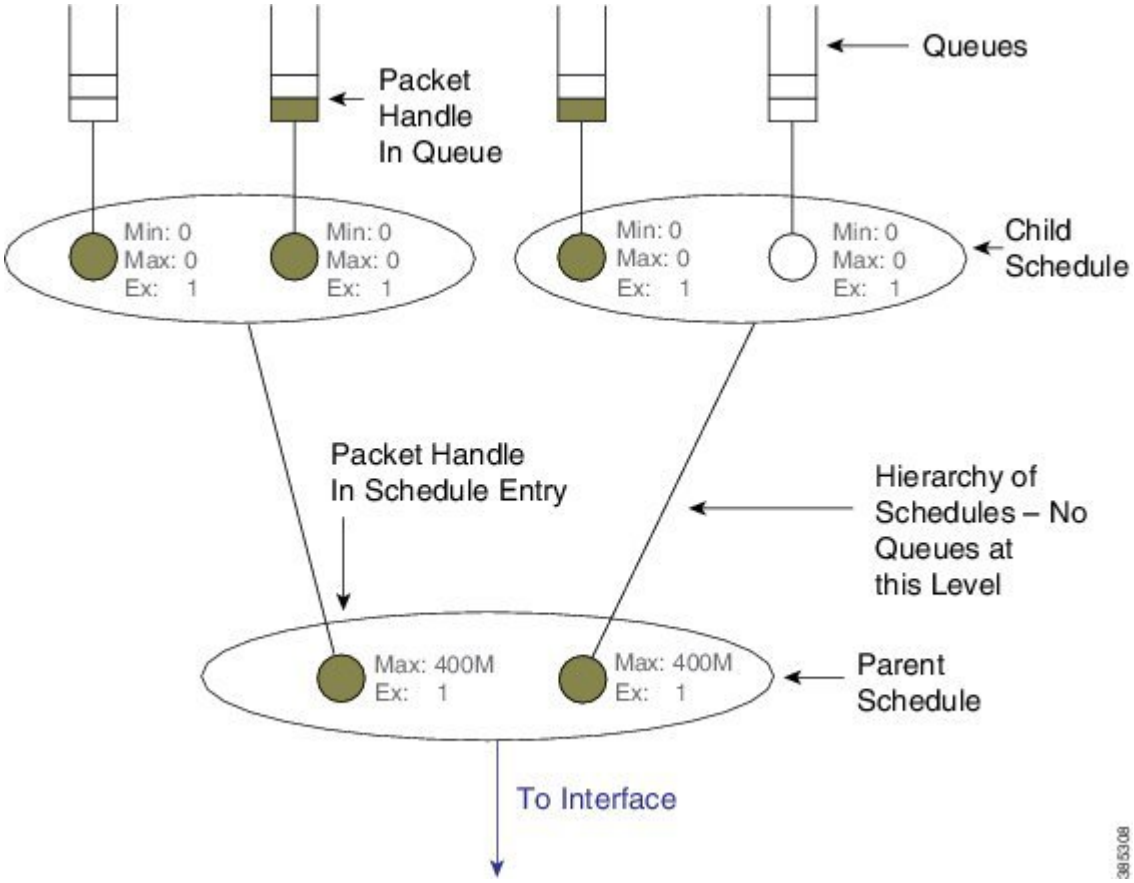
Definitions

We assume that you are now familiar with the role of a schedule and how a schedule entry contains information (packet handle, class queues, etc.) on how the child of that entry should be treated (see the Definitions, on page 31 discussion in the scheduling chapter). Here, we build upon that discussion.

The fundamental difference between what we show here and in the previous chapter is the *child schedule*, which may be a queue or another schedule. Hierarchical scheduling allows you to build complex structures with bandwidth sharing at multiple layers.

The following figure shows the basic hierarchical scheduling structure:

Figure 24: Hierarchical Scheduling Definitions



The first thing to notice from the diagram is that we implement a hierarchy of schedules and not a hierarchy of queues. This means that queues exist only at the *leaf layers* of the hierarchy and that packet handles (the packet representation vehicle) never move from queue to queue. Instead, a single packet handle is loaded into the *parent schedule* entry (provided a packet is waiting for transmission).

When detailing a scheduling hierarchy we describe schedules as parent or child (or indeed grandchild). These descriptions are relative. A parent schedule is one closer to the root of the hierarchy (closer to the interface). The child of a schedule could be either a schedule or a queue. We may also refer to schedules as a leaf or non-leaf schedule. A *leaf schedule* has solely queues as children; a *non-leaf schedule* will have at least one schedule as a child.

Looking at the diagram you can see that the schedule entry in the parent schedule (non-leaf)) has only two parameters per schedule entry. The Minimum Bandwidth parameter is only supported in leaf schedules – not in non-leaf schedules.

Scheduling Decisions - Root to Leaf

The following sequence of diagrams illustrates how the schedules in a hierarchy work in concert yet make local decisions when selecting the next packet to send through an interface.

Among the packets stored locally, the parent schedule will first decide on the most eligible packet to forward to the interface. After sending the associated packet handle, it will have a free spot in its own schedule entry – no packet handle from the child of that entry exists.

If the child is another schedule, the parent will send it a *pop* (a message that communicates "you pick your most eligible packet and send me that packet handle"). The child schedule will review the configuration of each entry, decide which packet should be sent next, and forward that packet handle to the parent schedule. The child will now have a free spot in its schedule entry. As the child is a queue no decision is necessary - the packet handle at the head of the queue will be loaded into the (child) schedule entry:

Figure 25: Scheduling Decision - Root to Leaf: Steps 1-2

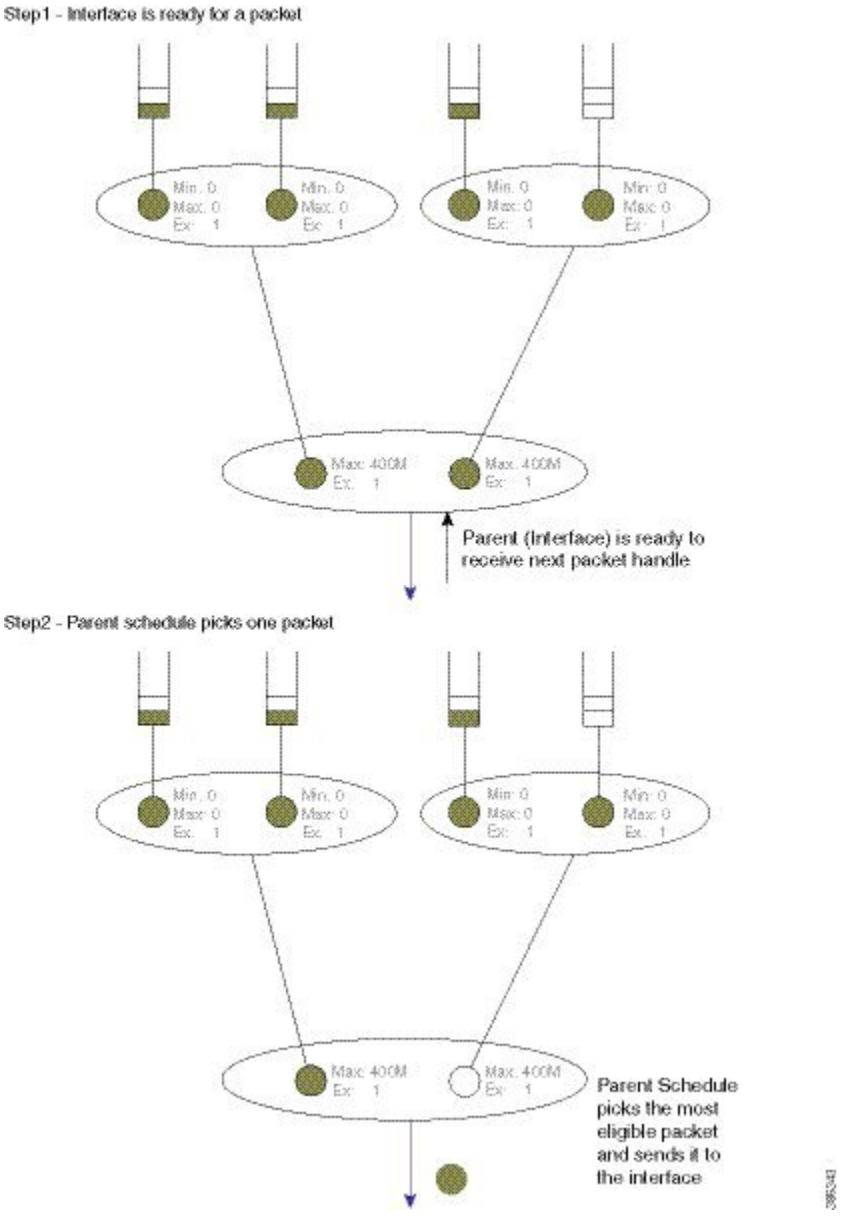


Figure 26: Scheduling Decision - Root to Leaf: Steps 3-4

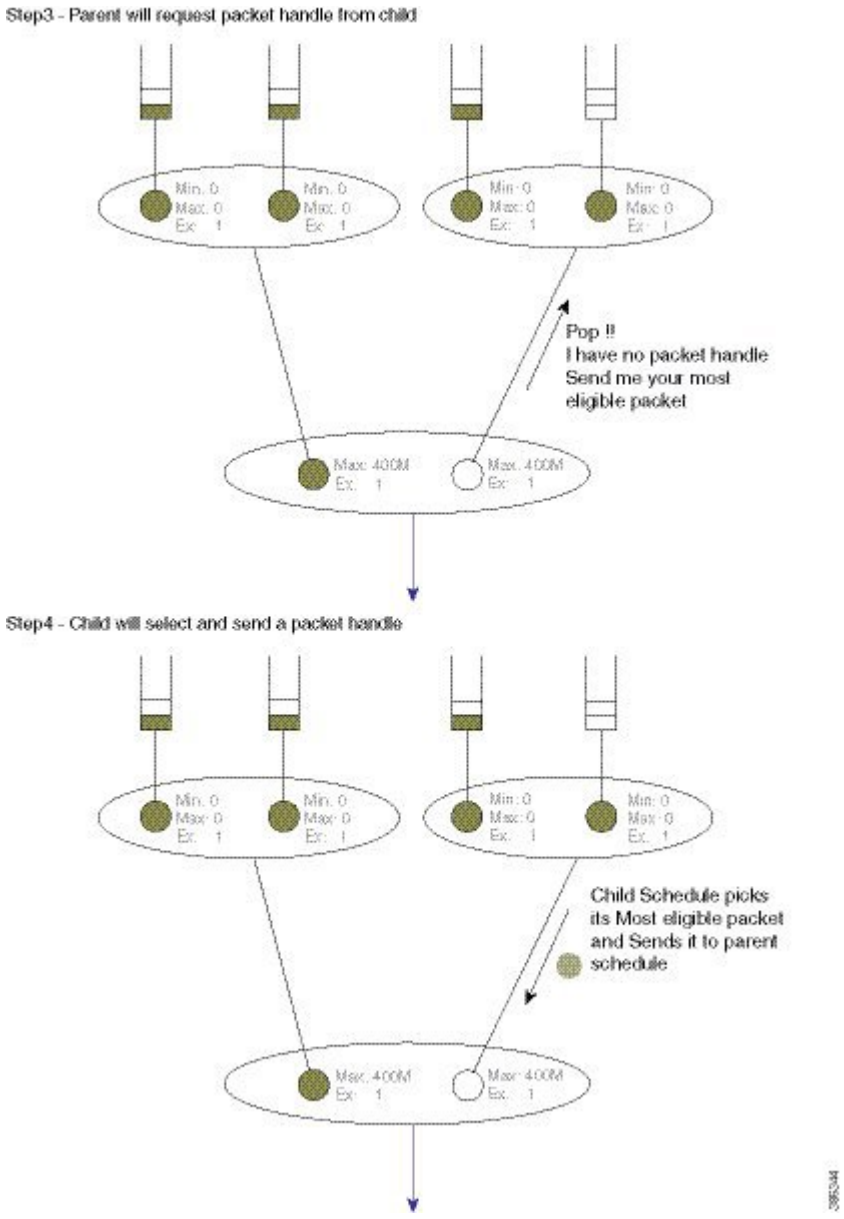
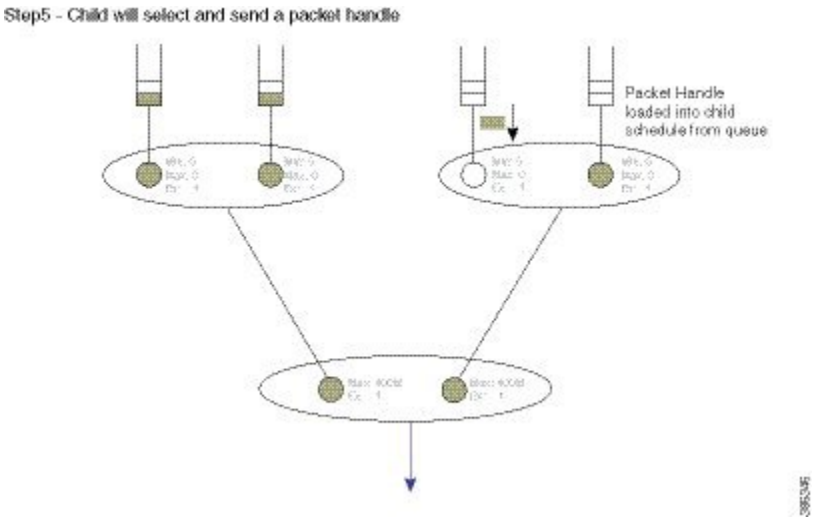


Figure 27: Scheduling Decision - Root to Leaf: Step 5

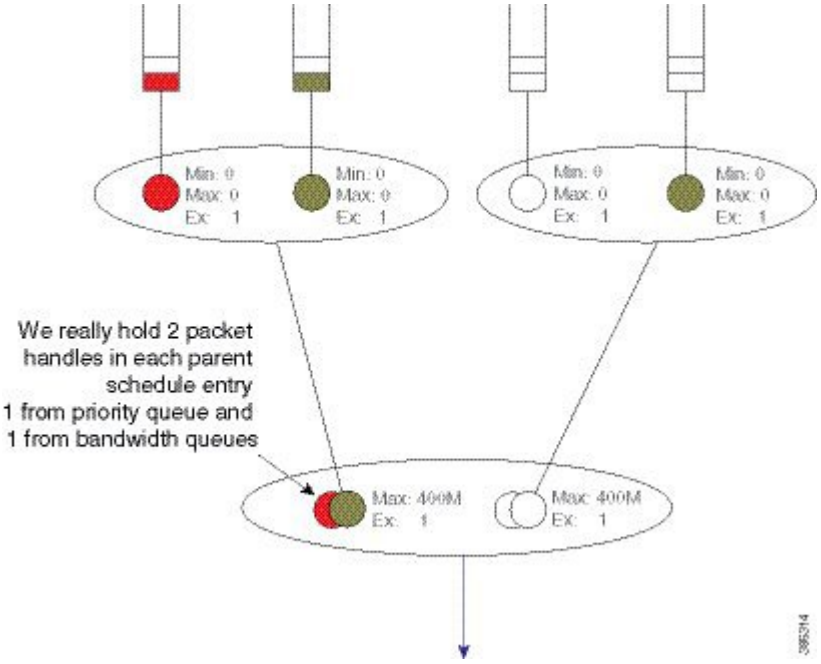


Concept of Priority Propagation

You will notice that thus far the descriptions have been somewhat simplistic in that they have only included bandwidth queues. In truth, for each child, a parent schedule can hold a *priority (queue) packet handle* and a *bandwidth (queue) packet handle* (we term this capability *passing lanes*). When a packet handle is sent from a child schedule to the parent we indicate whether it arose from a priority or a bandwidth class and we also indicate the priority level (we term this behavior *priority propagation*).

We will examine priority propagation later in this chapter. Here we merely introduce the concept so that the rules of hierarchical scheduling make sense:

Figure 28: Parent Schedule can hold Priority and Bandwidth Handles (Passing Lanes)



Observe in this hierarchy that priority service does not require configuration in the parent schedule entry; the (parent) schedule entry has only two parameters, Max and Excess Weight.

Hierarchical Scheduling Operation

In the scheduling chapter we describe how scheduling decisions are made for a flat policy attached to a physical interface. Here, we describe the scheduling rules for a leaf schedule (a scenario addressing a schedule with only queues as children). Those rules still hold.

We will now expand on that description to include the rules for the parent-child interaction:

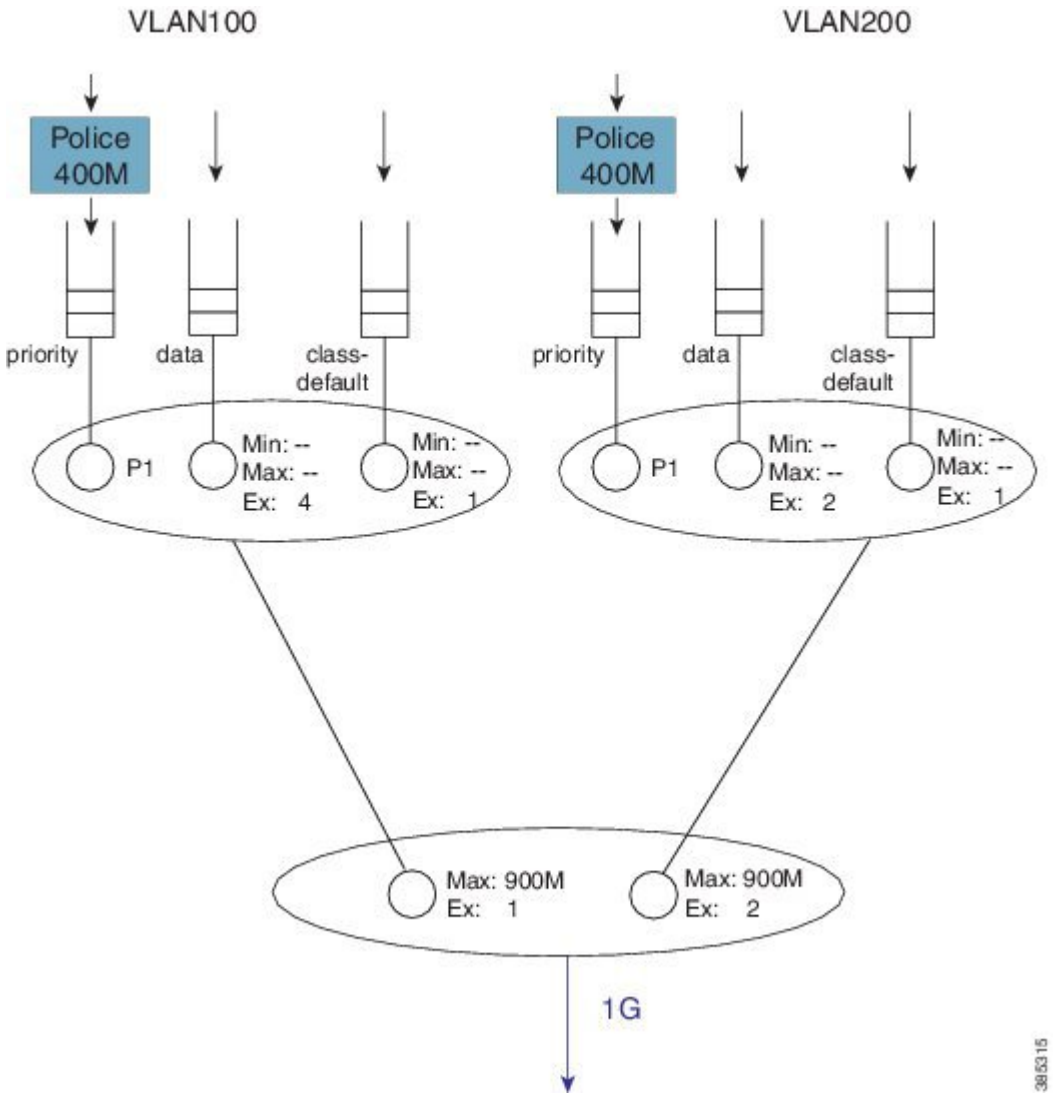
- Priority traffic counts towards Max (**shape** command) configured at the parent schedule.
- Priority traffic is unaltered by Ex (**bandwidth remaining ratio** command) configured at parent.
- Priority packets at the parent schedule will always be scheduled before bandwidth packets.
- Priority will be scheduled proportionally to the shape rate configured at parent. We include this point for completeness; it should not be a factor unless your priority load can oversubscribe the interface.
- Under priority propagation, a parent will know that a packet came from a priority queue but it will not know whether it was P1 (priority level 1) or P2 (priority level 2).
- Traffic from queues configured with the **bandwidth** or the **bandwidth remaining** commands are treated equally at the parent (*no min bandwidth propagation*). Henceforward, we refer to traffic from any bandwidth queue as *bandwidth traffic*.
- Excess weight configuration at the parent controls the fairness between bandwidth traffic from multiple children competing for any physical bandwidth not consumed by priority traffic.

To understand these rules, let's look at the following configuration example. Later, we will detail how a configuration is mapped into a datapath configuration. For now, the diagram and schedule entries shown in the diagram are sufficient to understand the behavior:

```
policy-map child100
  class priority
    priority
    police cir 400m
  class data
    bandwidth remaining ratio 4
!
policy-map parent100
  class class-default
    shape average 900m
    service-policy child100
!
policy-map child200
  class priority
    priority
    police cir 400m
  class data
    bandwidth remaining ratio 2
!
policy-map parent200
  class class-default
    shape average 900m
    bandwidth remaining ratio 2
    service-policy child200
!
int g1/0/4.100
  encaps dot1q 100
  service-policy out parent100
!
int g1/0/4.200
  encaps dot1q 200
  service-policy out parent200
```

The following diagram shows the scheduling hierarchy associated with the previous configuration. As described previously, the **shape** command in the parent policy(s) sets the Max parameter and the **bandwidth remaining ratio** command sets the Ex parameter in the schedule entry (rules 1 and 2). The latter defaults to 1 if not explicitly set:

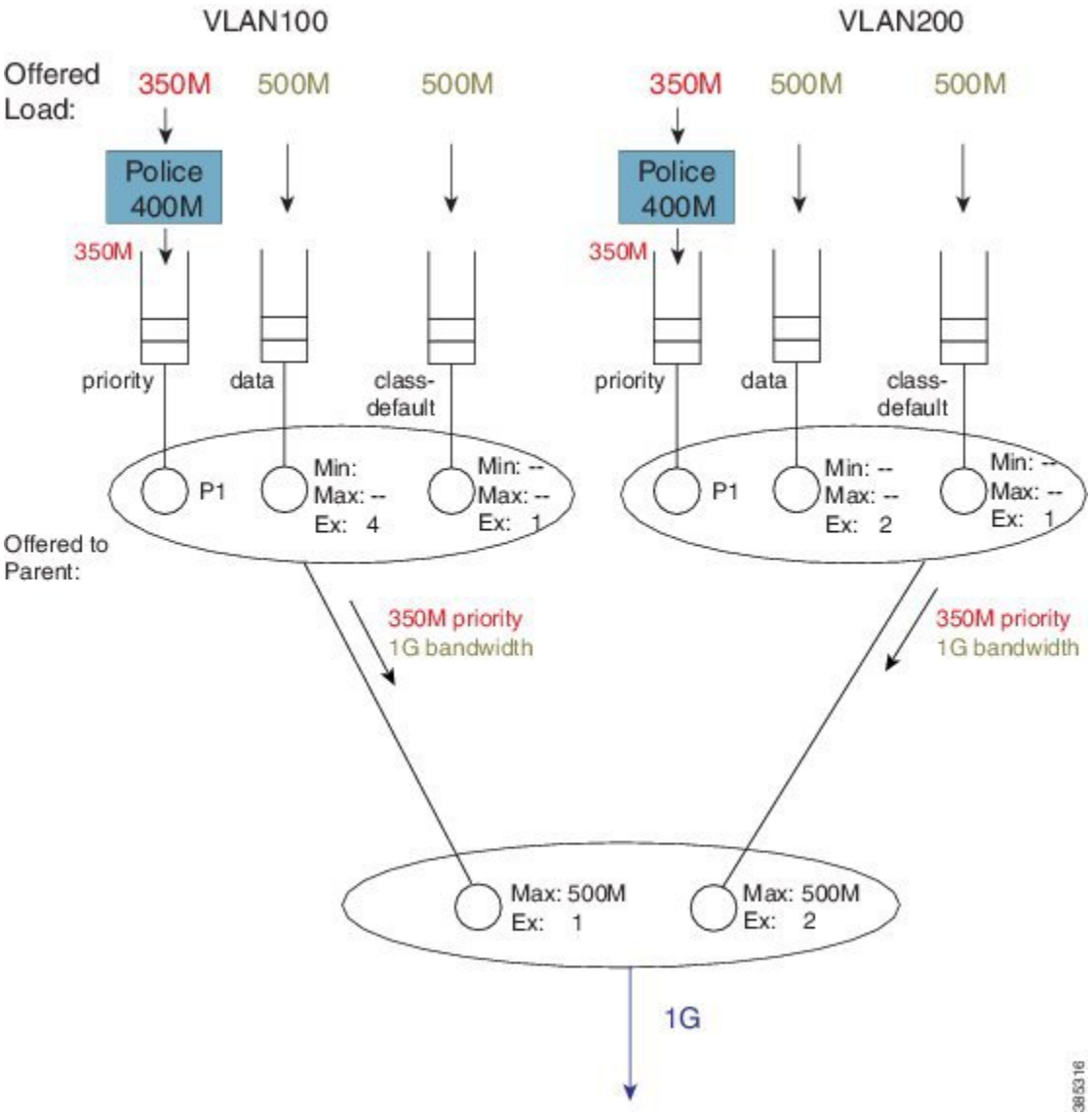
Figure 29: Scheduling Hierarchy Example - Forwarding the Entire Offered Priority Load



Let's now look at the expected throughput for an offered load.

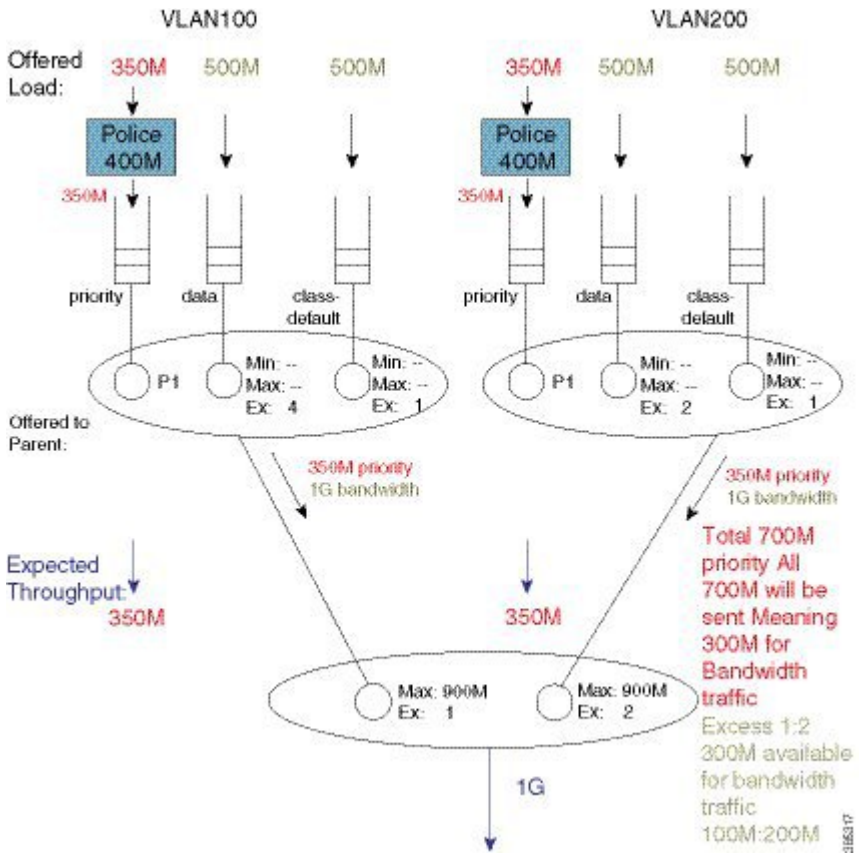
Note The following examples ignore overhead accounting – they are intended solely to illustrate how to calculate expected throughput independent of minor details.

Figure 30: Calculating What is Offered to the Parent from Each Child Schedule



In calculating expected throughput, the first step is to eyeball the offered load per class. The next step is to aggregate them and observe the total loads from priority and bandwidth classes that will be offered to the parent:

Figure 31: Calculating the Remaining Bandwidth for Bandwidth Queues



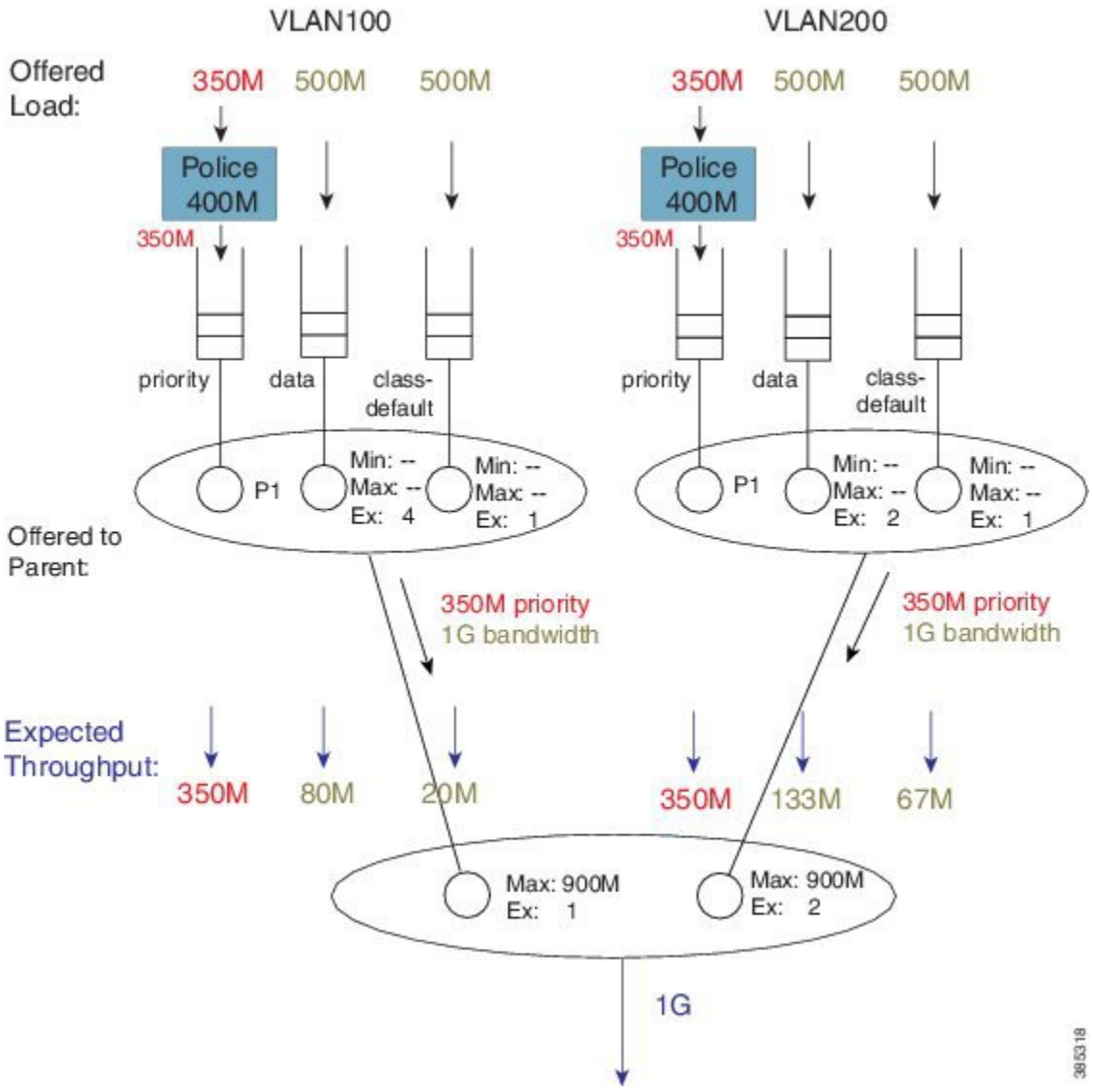
Each child schedule is offering 350 Mbps of priority traffic to the parent. Because the interface has 1 Gbps of available bandwidth it will forward the entire 700 Mbps offered priority load.

According to rule 3, we schedule priority traffic before bandwidth traffic. As the Max (rate) for each parent schedule entry exceeds the offered priority load from that entry's child schedule, we forward the entire 350 Mbps of traffic.

With the scheduled priority load (350 Mbps + 350 Mbps of traffic), we can now calculate the (remaining) bandwidth for bandwidth (queue) traffic (300 Mbps or 1 Gbps of total bandwidth available - 700 Mbps consumed by priority load).

The parent schedule will use the Ex configuration to apportion the 300 Mbps (remaining) bandwidth. With Ex values of 1 and 2, for VLAN100 and VLAN200, respectfully, the bandwidth will be shared 1:2. VLAN100 will receive 100 Mbps and VLAN200 will receive 200 Mbps of bandwidth traffic throughput:

Figure 32: Bandwidth Sharing based on the Excess Weights in the Child Schedule



To calculate how this 100 Mbps will be apportioned, we can now examine the bandwidth queue's schedule entries (in the schedule) for VLAN100.

No Min guarantees are configured (the **bandwidth** command is not supported in parent schedules), so all sharing hinges on the scheduled Ex values in the child schedule. Based on the settings (4 for class data and 1 for class class-default) the 100 Mbps will be shared 4:1 (class data receives 80 Mbps; class class-default receives 20 Mbps).

If we follow the same approach for VLAN200, the 200 Mbps available is split 2:1. Class data will receive 133 Mbps and class class-default will receive 67 Mbps.

You probably noticed that every class was oversubscribed. This means the expected throughput we calculated was also the minimum guaranteed service rate for each class. Under hierarchical scheduling, bandwidth sharing at the parent schedule ensures that we don’t waste bandwidth if any child schedule does not have packets

waiting for transmission. Similar to bandwidth sharing in flat policies, bandwidth unused by one child is available to others.

Priority Propagation

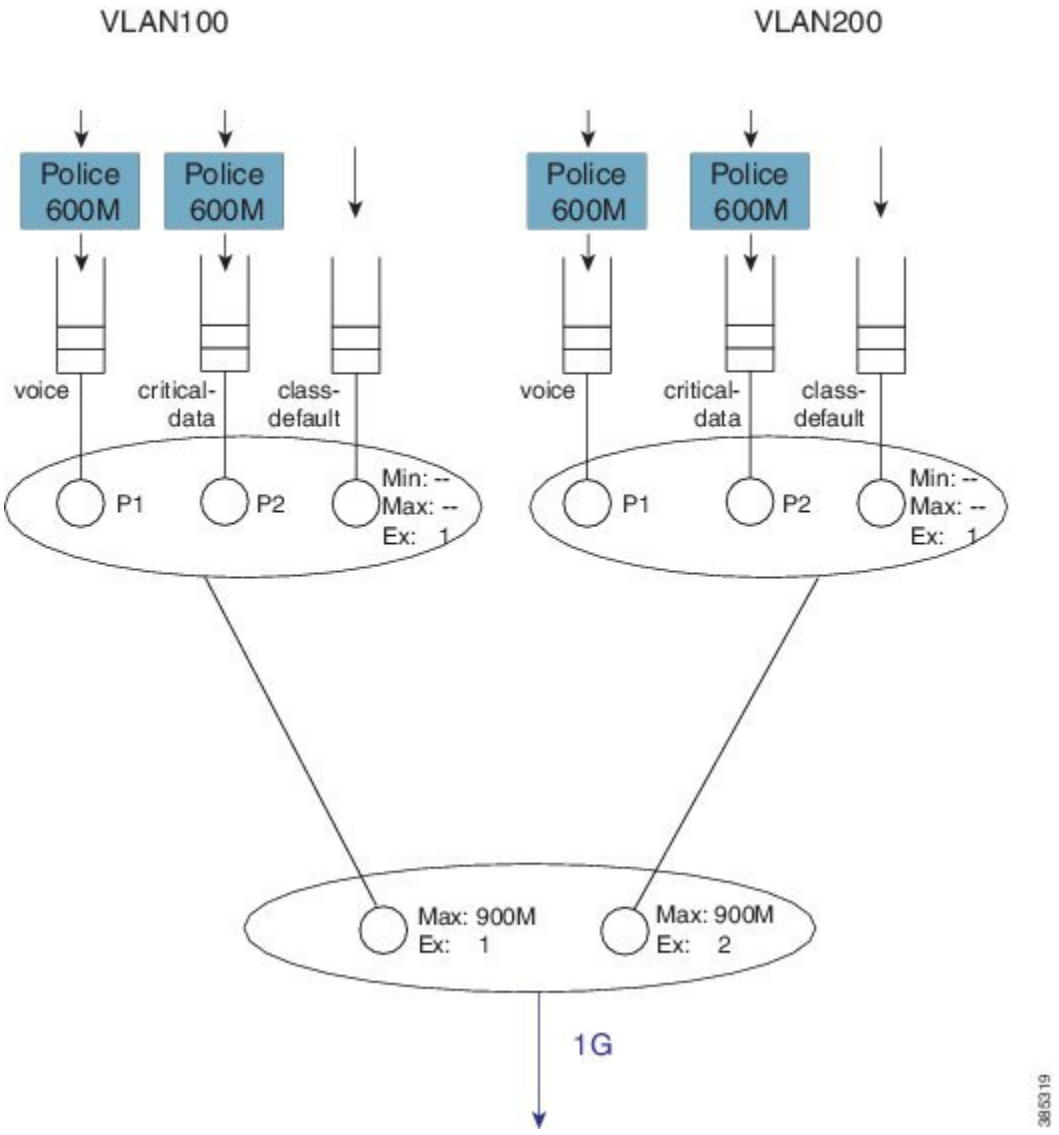
Regarding the Concept of Priority Propagation, on page 83, we will now use the following sample configuration to highlight a few points:

```
policy-map child
  class voice
    priority
    police cir 600m
  class video
    priority
    police cir 600m
!
policy-map parent100
  class class-default
    shape average 900m
    service-policy child
!
policy-map parent200
  class class-default
    shape average 900m
    bandwidth remaining ratio 2
    service-policy child
!
int g1/0/4.100
  encaps dot1q 100
  service-policy out parent100
!
int g1/0/4.200
  encaps dot1q 200
  service-policy out parent200
```

Note We are using the same child policy in both parent policy-maps. Unique policy-maps are unnecessary at any level; if the requirements match, you can share child or even parent policy-maps.

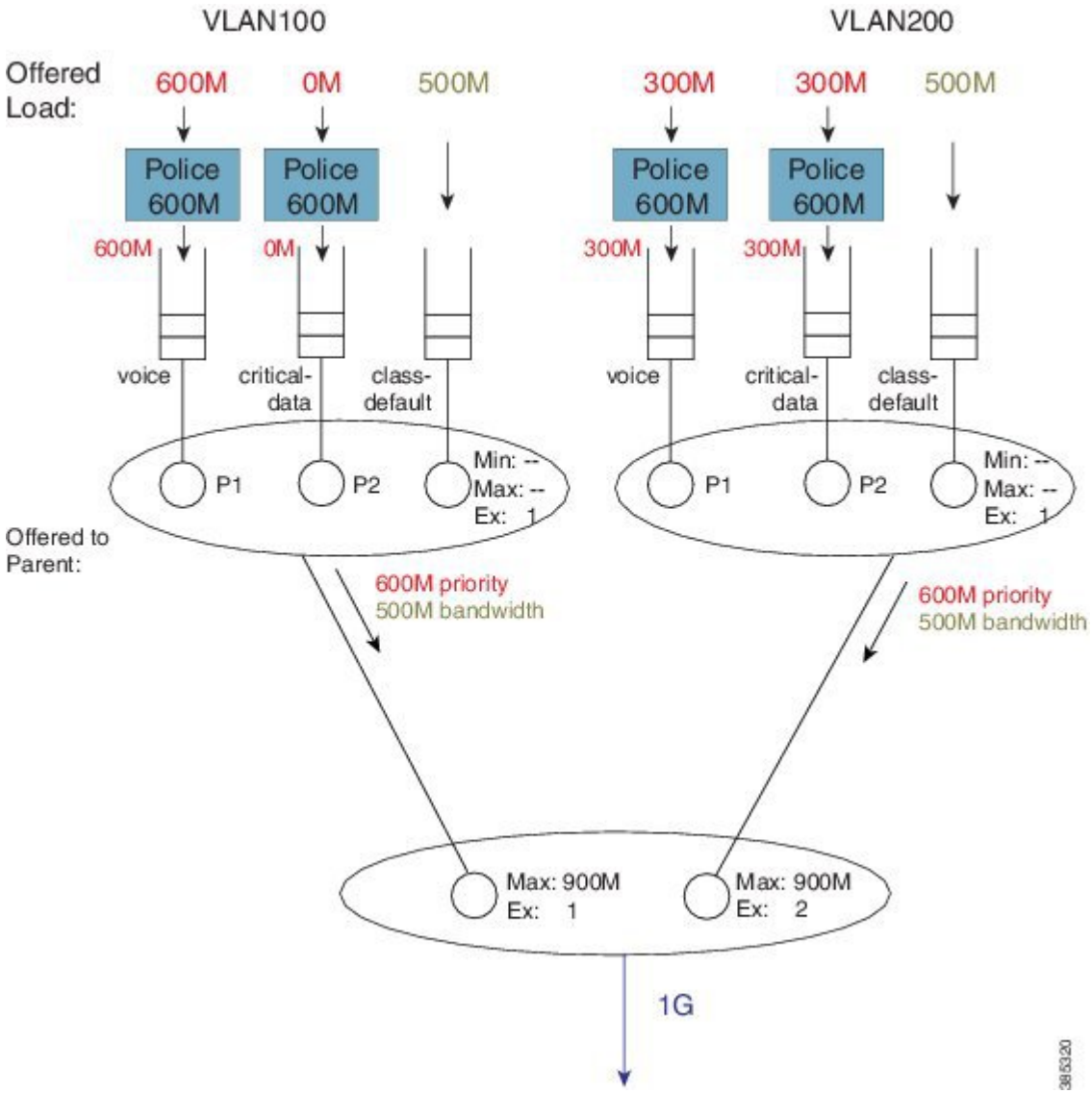
A hierarchy created for this configuration would look as follows:

Figure 33: Scheduling Hierarchy Example - Multi-level Priority Queuing in Child Schedule



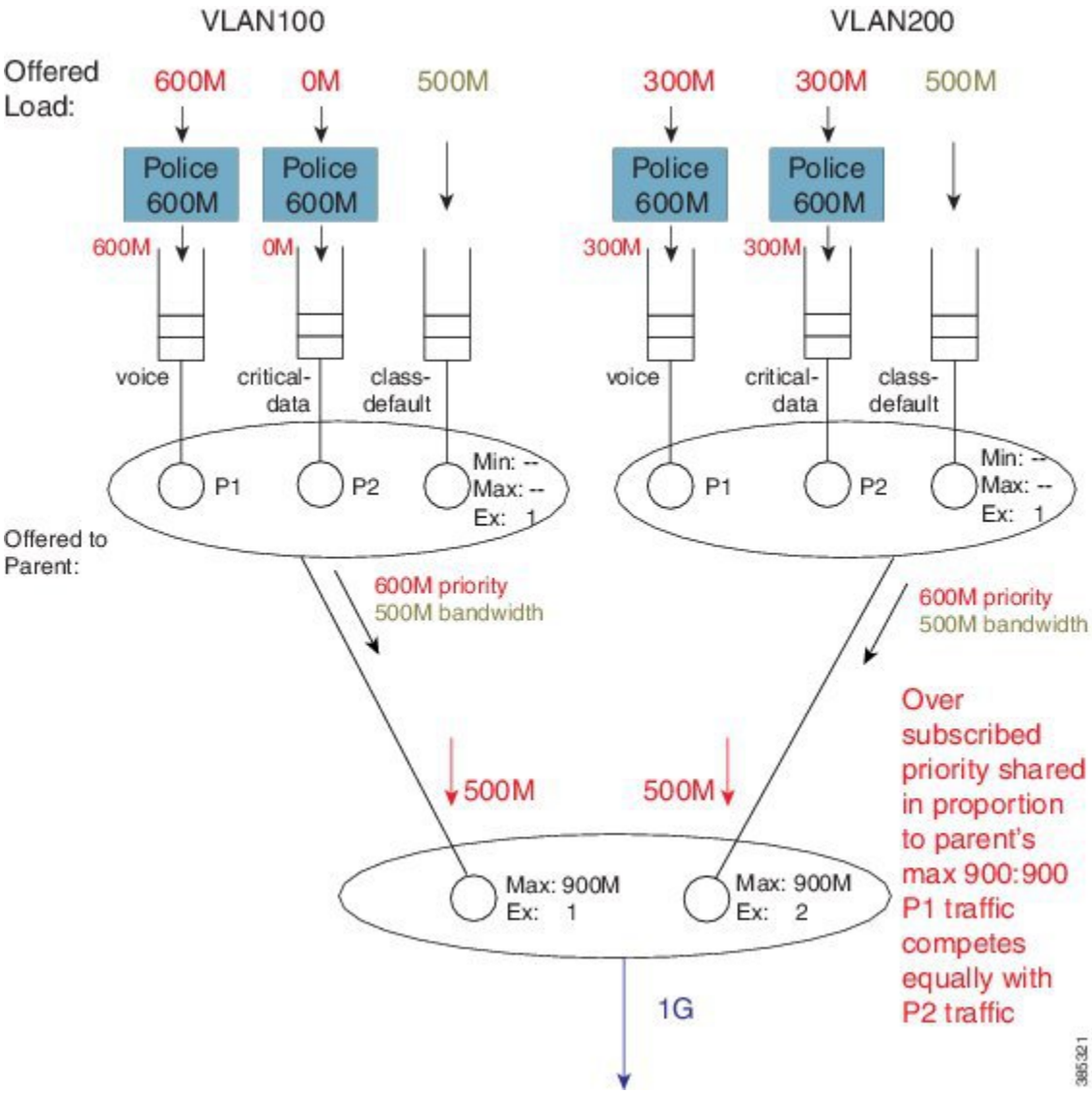
The scenario differs from that in the Concept of Priority Propagation, on page 83. We now have multi-level priority queuing in the child schedule (e.g., P1 [priority level 1] and P2 [priority level 2] classes). The following diagram shows the load offered to each class:

Figure 34: Multi-level Priority Queuing - Load Offered to each Class



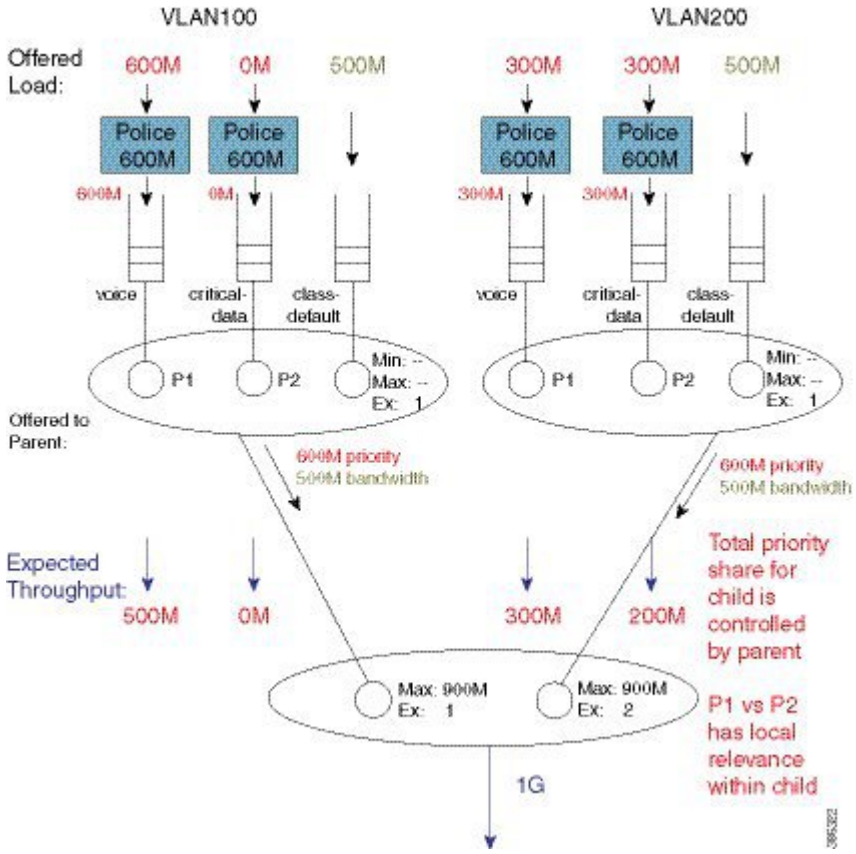
Now let's look at the total load offered from the priority and bandwidth queues (for each child) to the parent:

Figure 35: Oversubscribed Priority Queues shared relative to the Parent's Max ratio



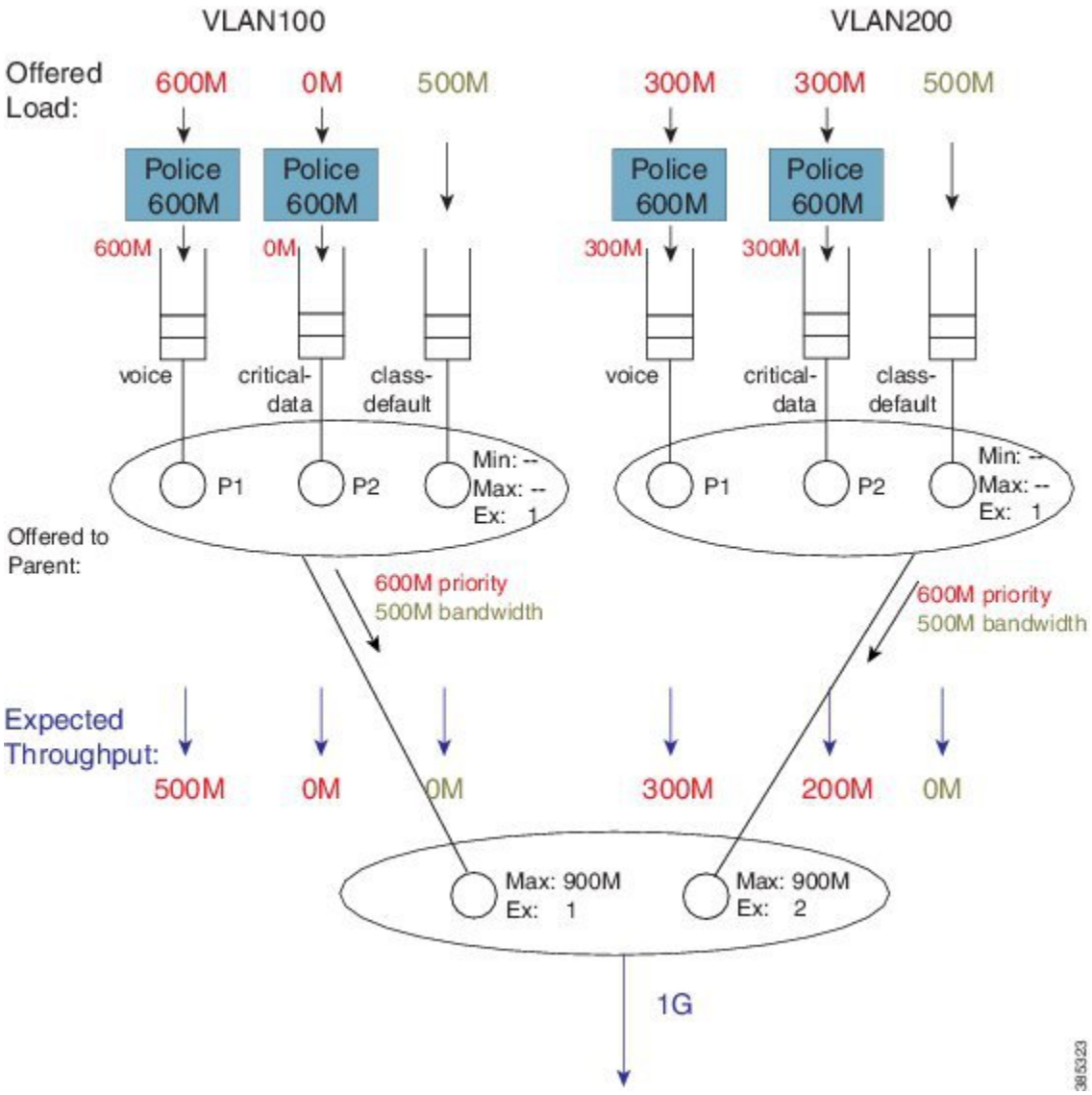
In rule 4 (see Hierarchical Scheduling Operation, on page 84) we stipulated that a parent will schedule an offered priority load *proportional to the shape rate configured in its schedule entry*. Here, each child has a Max rate ("shape" in the parent policy) of 900 Mbps and offers 600M priority traffic (i.e. 1.2 Gbps [600M + 300M + 300M traffic] when only 1 Gbps is available). The parent schedule will apportion 500 Mbps to each child. The key point to note here is that P1 from VLAN100 competes equally with P2 traffic from VLAN200. (Recall from rule 5 that priority propagation alerts the parent that a packet arose from a priority queue but does not indicate the priority level.)

Figure 36: Parent Controls Total Priority Share for Child



The parent schedule accepts 500 Mbps of priority load from VLAN200. The child schedule is responsible for apportioning bandwidth within that 500 Mbps. The child policy has P1 configured in the voice class, which means that the child schedule will always pick packets from that queue first (i.e., priority levels have local significance within a schedule). The expected throughput for the voice class in VLAN200 is 300 Mbps. The class critical-data will receive 200 Mbps (the unused share of the 500 Mbps – 300 Mbps in this example):

Figure 37: Child Schedule Apportions Bandwidth received from Parent Schedule



What about the expected throughput from the bandwidth queues? As the offered priority load exceeded the physical bandwidth available, nothing remained for the bandwidth queues. This example effectively highlights that priority classes can starve bandwidth queues completely. If control packets are not in priority queues, you might experience network instability. In fact, failure to place control packets in priority queues could be considered a misconfiguration!

Note Ensure that the physical bandwidth available exceeds the sum of all priority class policers, so that the latter can't starve others of service.

Please be aware that the concept of priority propagation does not end in the scheduling hierarchy. When we mark a packet as stemming from a priority class, that tag is carried to the egress interface. In egress carrier or

interface cards, we find multiple places where passing lanes enable priority packets to arrive at the interface as quickly as possible.

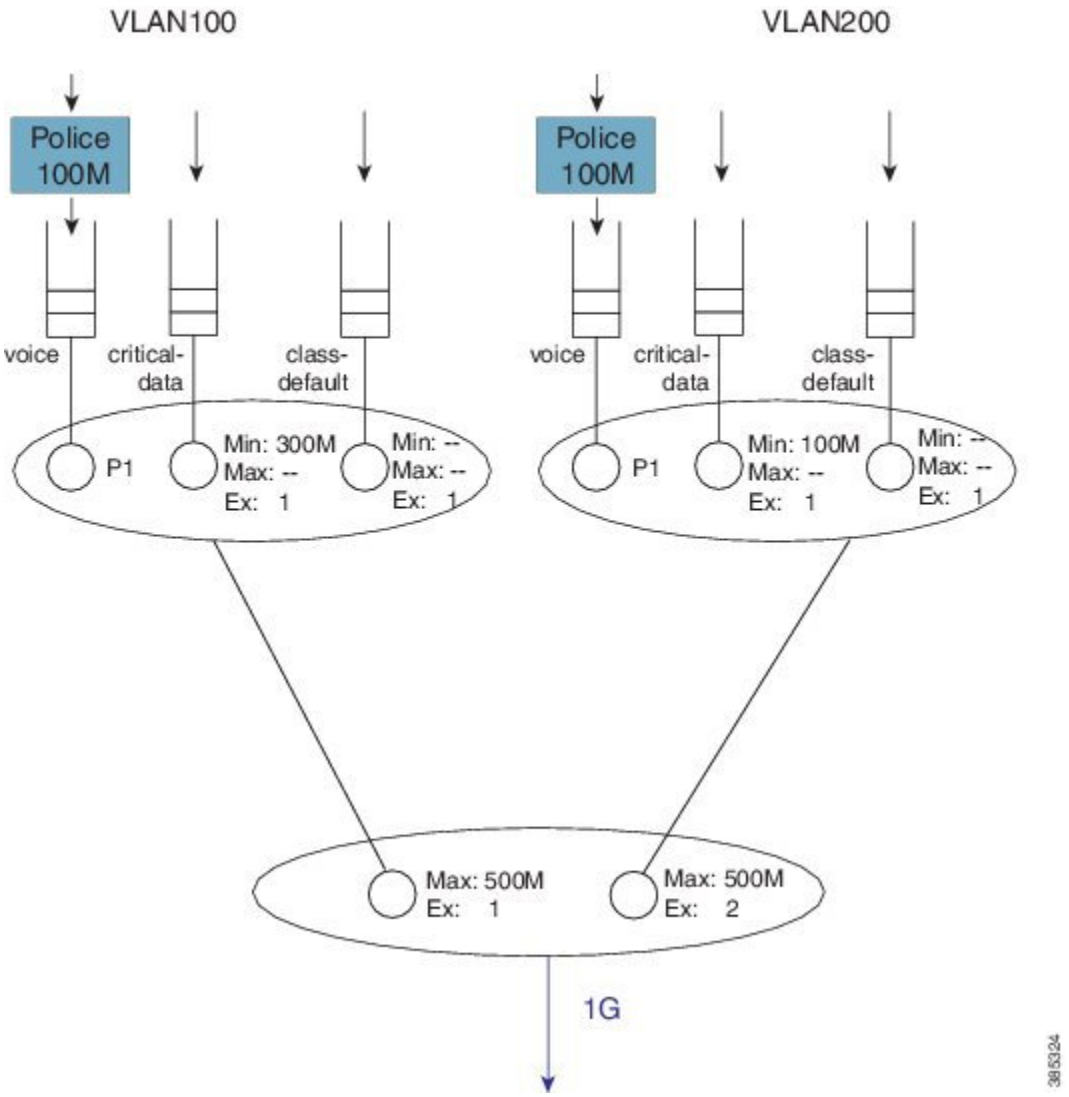
Bandwidth Command in Leaf Schedules

We have already stated that although the **bandwidth** command is not supported in parent schedules (and so a Min setting is absent), it is supported in leaf schedules. With the following configuration, we will explain the operation of the **bandwidth** command in a child policy-map. (Lines flagged with asterisks indicate how this configuration compares with that presented in priority propagation.

```
policy-map child100
  class voice
    priority
    police cir 100m
  class critical-data
    bandwidth 300000                  ****
!
policy-map child200
  class voice
    priority
    police cir 100m
  class critical-data
    bandwidth 100000                 ****
!
policy-map parent100
  class class-default
    shape average 500m
    service-policy child100
!
policy-map parent200
  class class-default
    shape average 500m
    bandwidth remaining ratio 2
    service-policy child200
!
int g1/0/4.100
  encaps dot1q 100
  service-policy out parent100
!
int g1/0/4.200
  encaps dot1q 200
  service-policy out parent200
```

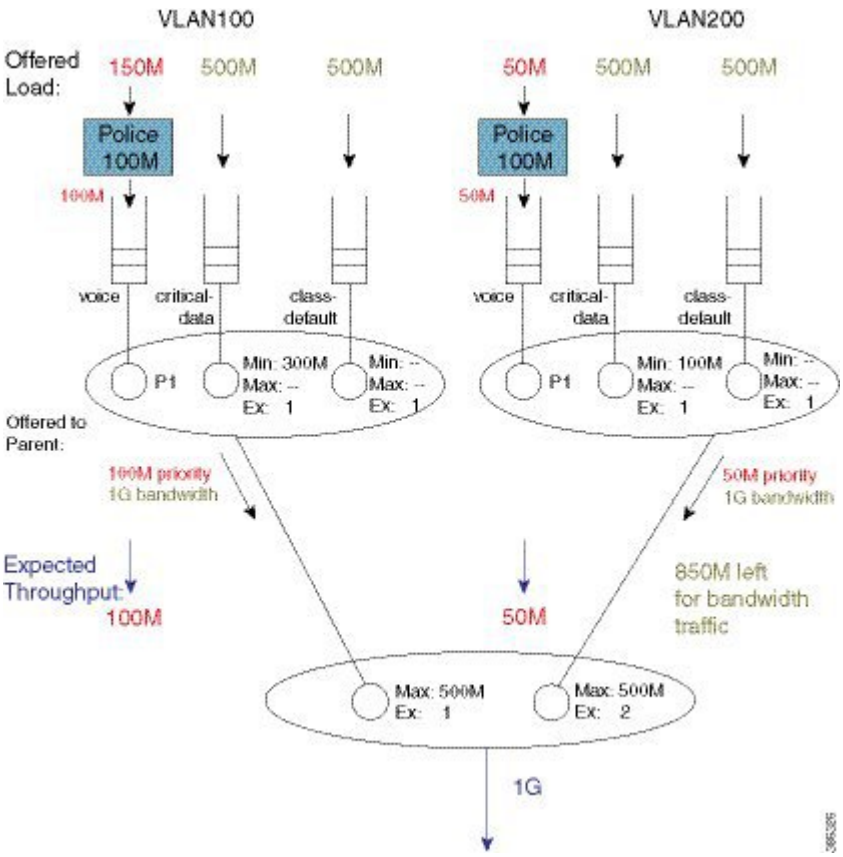
A hierarchy created for this configuration would look as follows:

Figure 38: Scheduling Hierarchy Example - Bandwidth Command Application in Leaf Schedules



To explain the operation of this hierarchy let's consider the following offered loads (to each class):

Figure 39: Load Offered to each Class

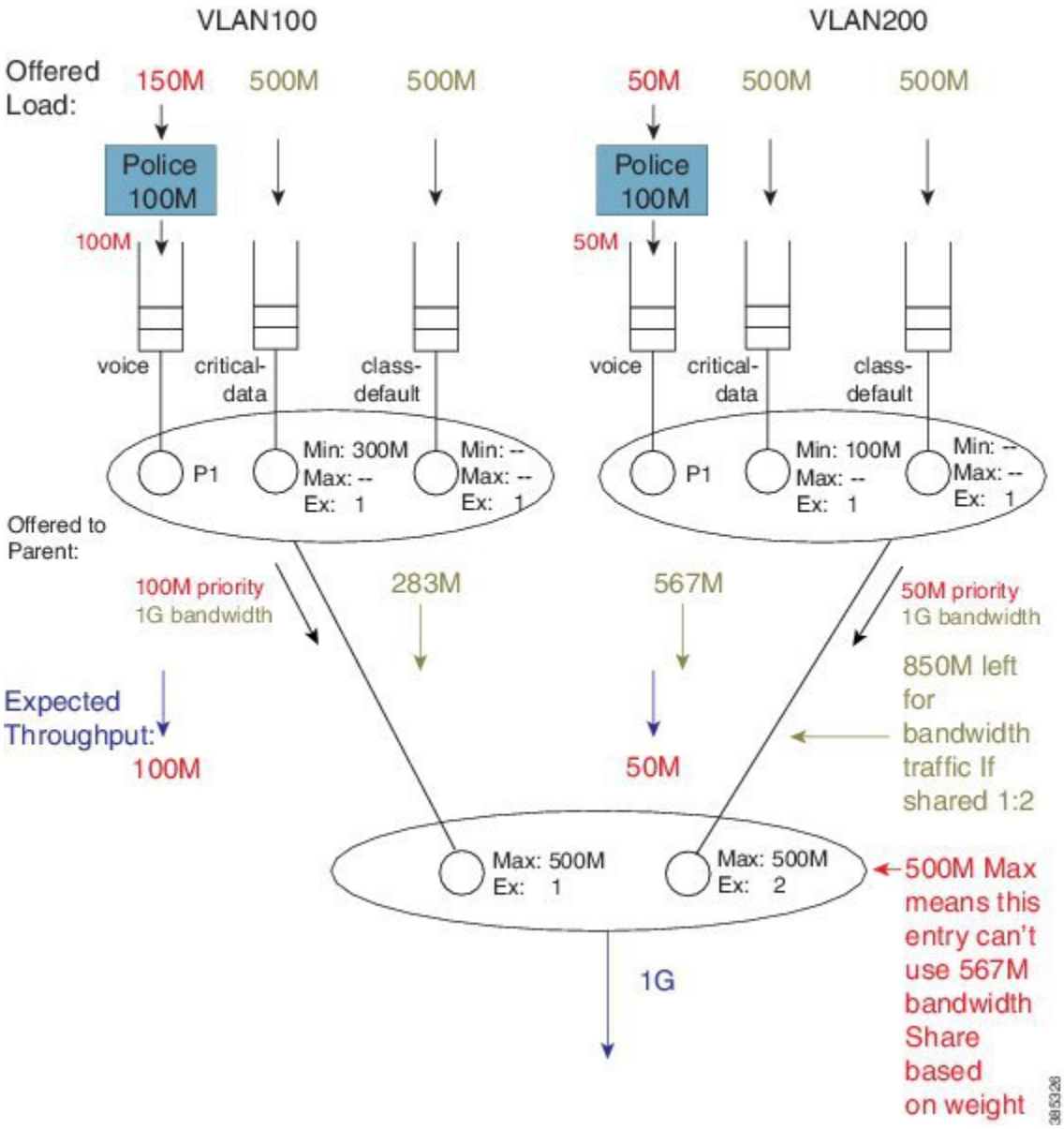


As with the previous example, we first examine the total load offered from the priority and bandwidth queues for each child to the parent.

The total priority load in this example is 150M. Each child is offering less than their Max rate (shape in parent policy) and the aggregate offered-priority load is less than the 1 Gbps total available bandwidth. (Recall the example in schedule operation where the total offered priority traffic exceeded the total available bandwidth.) This means the entire priority load offered from each child would be forwarded. With 150 Mbps scheduled from priority queues, we have 850 Mbps available for bandwidth queues.

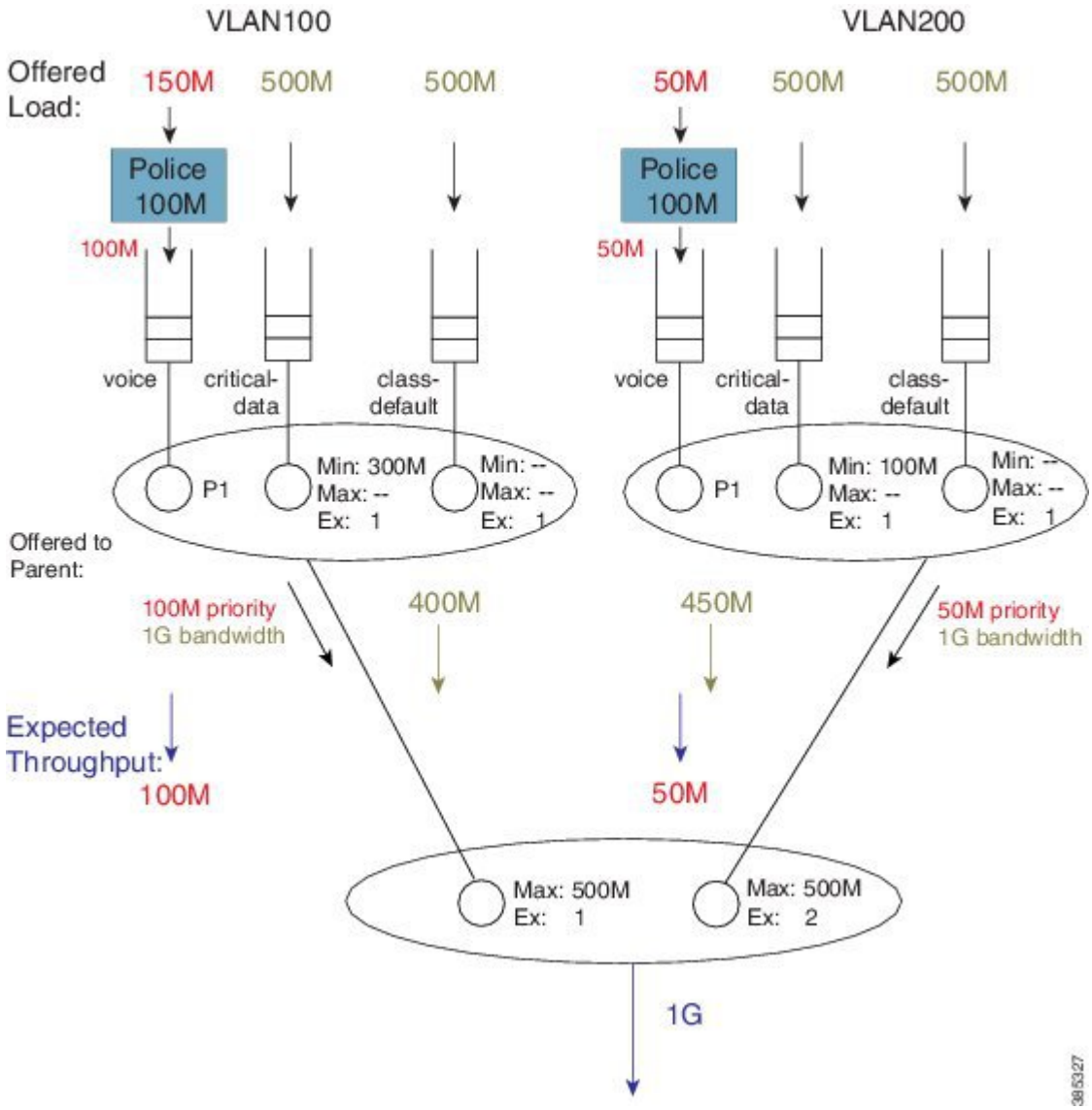
To calculate how to apportion the bandwidth between each child, let's first look at the excess weight configured in each schedule entry in the parent:

Figure 40: Apportioning Bandwidth Share Between Children



If we focus exclusively on the excess weight, VLAN200 would be apportioned 567 Mbps of the interface bandwidth (2/3 of 850 Mbps). However, we also need to factor in the Max value (500 Mbps) configured in the schedule entry, which includes the 50 Mbps of priority traffic from that child. This means that VLAN 200 will actually forward 450 Mbps of bandwidth traffic and VLAN100 will forward 400 Mbps of bandwidth traffic (850 Mbps - 450 Mbps for VLAN 200):

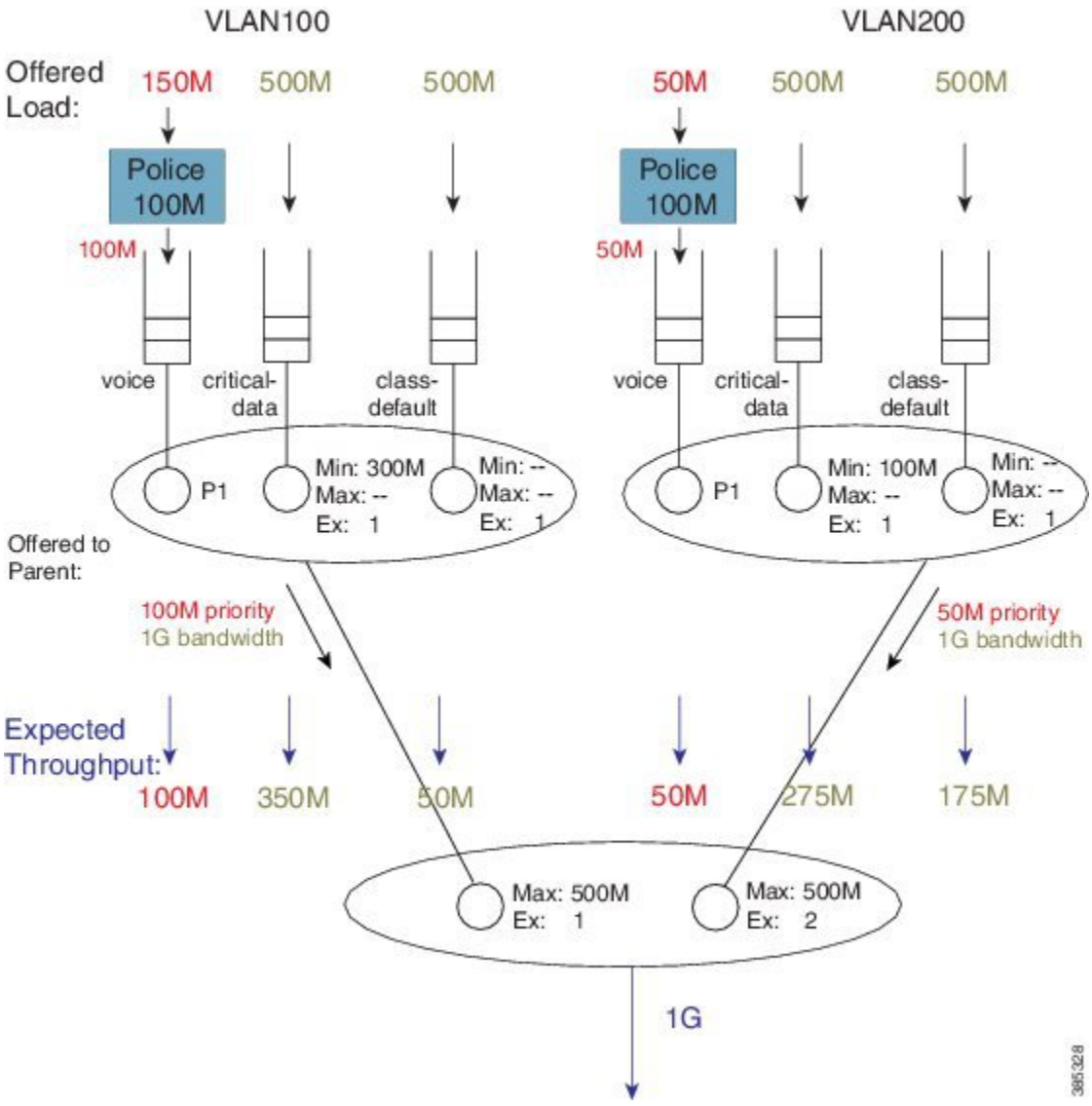
Figure 41: How the Max value of the Parent's Schedule Entry influences Bandwidth Sharing



As the sum of the Max values at the parent level is less than or equal to the available physical bandwidth, the Ex values in the parent policies do not add value – each child will receive a total throughput matching its shape rate (e.g., for VLAN100, 100M + 400M = 500M [the shape rate]). Observe that with such a configuration, any bandwidth unused by one child would not be available to another. Any child is always limited to the configured Max value.

With the total throughput for bandwidth classes in each child, we can now calculate the throughput each individual class in that child will receive. Recall from the schedule operation that Min bandwidth guarantees are always serviced first and any excess bandwidth is shared based on the Ex values, which always default to 1:

Figure 42: Factoring Total Throughput to Apportion Bandwidth within each Child Schedule



For example, the bandwidth apportioned to the class critical-data of VLAN200 would be 275M (100M (Min guarantee) + ½ (450M -100M), where we derive "½" from the Ex ratio of 1:1).

Bandwidth Command is Only Locally Significant

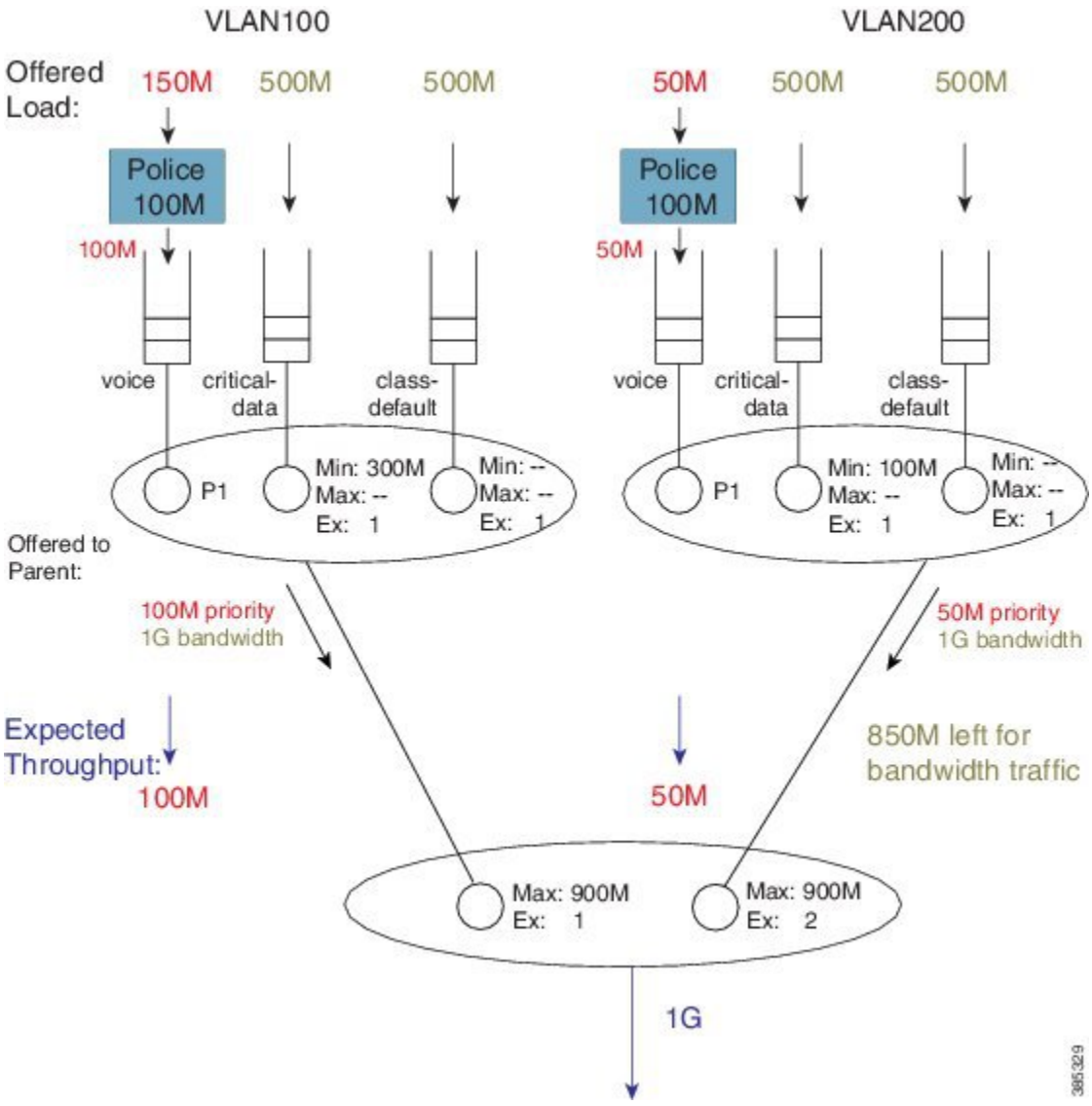
To highlight the risk of using the **bandwidth** command in hierarchical policies, we will modify the previous configuration example by increasing the parent shapers so that they are no longer the constraining factor. In the revised configuration, the sum of the parent shapers oversubscribes the physical bandwidth available. (Commands flagged with asterisks indicate how this configuration differs from that presented in Bandwidth Command in Leaf Schedules, on page 96.)

```
policy-map child100
  class voice
```

```
    priority
    police cir 100m
  class critical-data
    bandwidth 300000
!
policy-map child200
  class voice
    priority
    police cir 100m
  class critical-data
    bandwidth 100000
!
policy-map parent100
  class class-default
    shape average 900m                          ****
    service-policy child100
!
policy-map parent200
  class class-default
    shape average 900m                          ****
    bandwidth remaining ratio 2
    service-policy child200
!
int g1/0/4.100
  encaps dot1q 100
  service-policy out parent100
!
int g1/0/4.200
  encaps dot1q 200
  service-policy out parent200
```

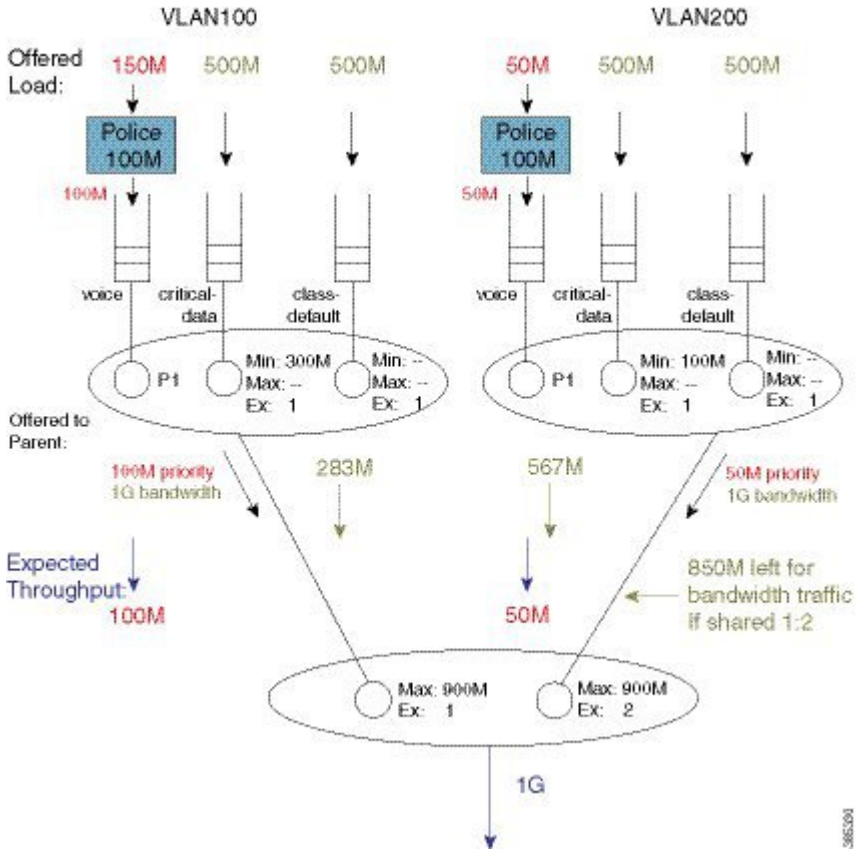
If we apply the offered load profile from Bandwidth Command in Leaf Schedules, on page 96, the hierarchy and load profile will appear as follows:

Figure 43: Scheduling Hierarchy Example - Parent Shapers no Longer Constraining



Similar to the previous example, 850 Mbps are available (remaining) for bandwidth queues. (Inspecting the sum of priority load and bandwidth traffic share for each child, you notice that the Max value in each parent schedule would not be exceeded.) Based on the excess weights configured in the parent schedule, we calculate the bandwidth share each child would receive (from the parent schedule): 283 Mbps and 567 Mbps.

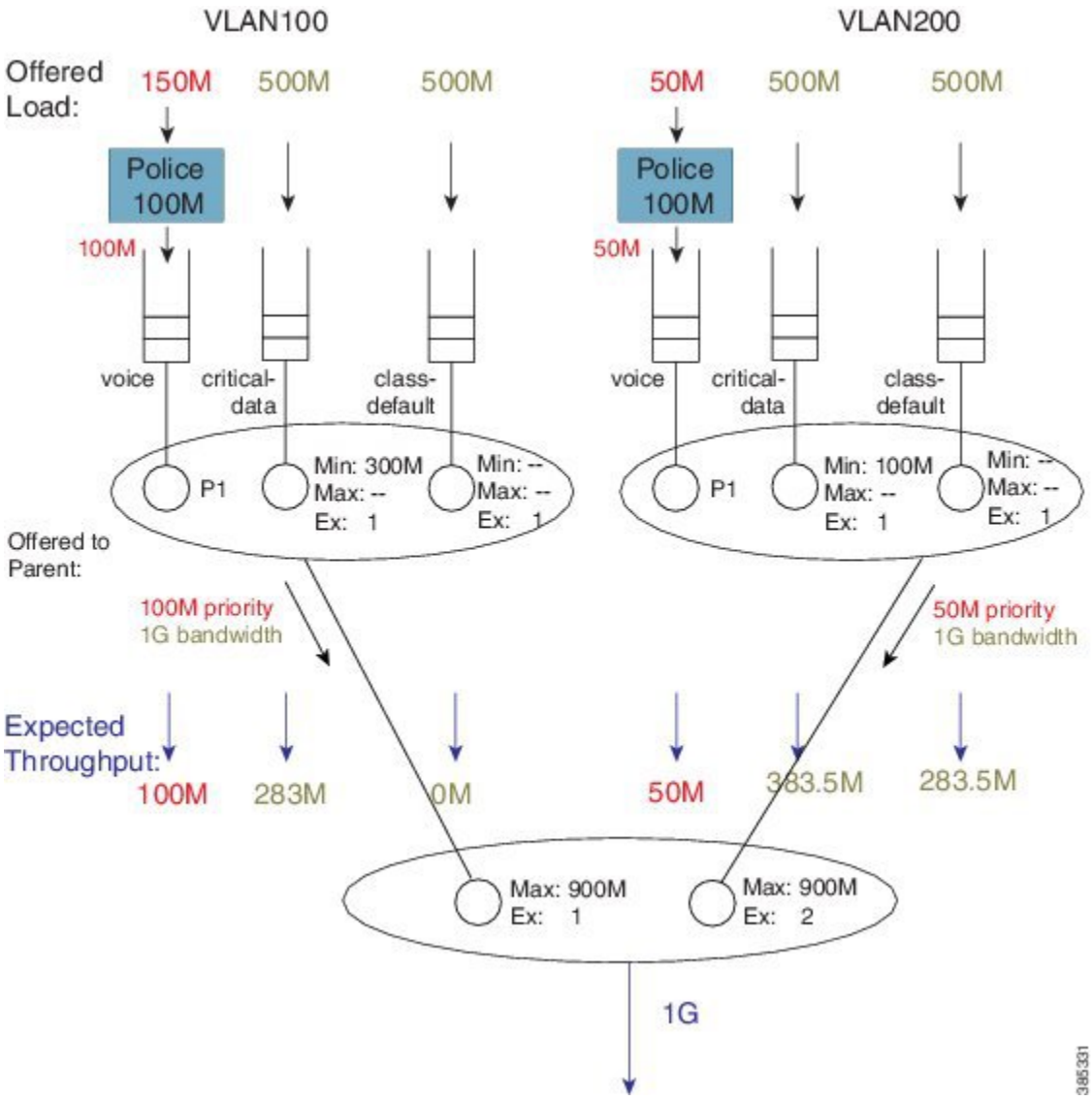
Figure 44: Calculating Bandwidth Share for Each Child based on Excess Weight Configured in Parent Schedule



Note In contrast to the previous example, because the shape values are no longer constraining, total throughput for each child does not match the shape rate.

Let's examine the entries in each child schedule to see how bandwidth would be apportioned to each class:

Figure 45: How Child Entries Dictate how Bandwidtth is Apportioned



Viewing the child schedule for VLAN100, you notice that the schedule entry for class critical-data has a Min value of 300 Mbps configured. The 283 Mbps bandwidth apportioned to this schedule is insufficient to satisfy this guarantee.

The key point of this discussion is that Min bandwidth guarantees are only locally relevant; Min bandwidth propagation does not exist. Traffic from one child schedule competes equally with excess traffic from another.

Also, please note that using Min in scheduling hierarchies could starve other classes of service (in this example, class-default in VLAN100). To avoid this, use only the **bandwidth remaining** command in child policies.

Tip

If you oversubscribe parent shapers in a hierarchical policy and want to avoid starving some classes of service, ensure that the sum of policers on your priority queues does not exceed the bandwidth available. Furthermore, consider using the **bandwidth remaining** over the **bandwidth** command.

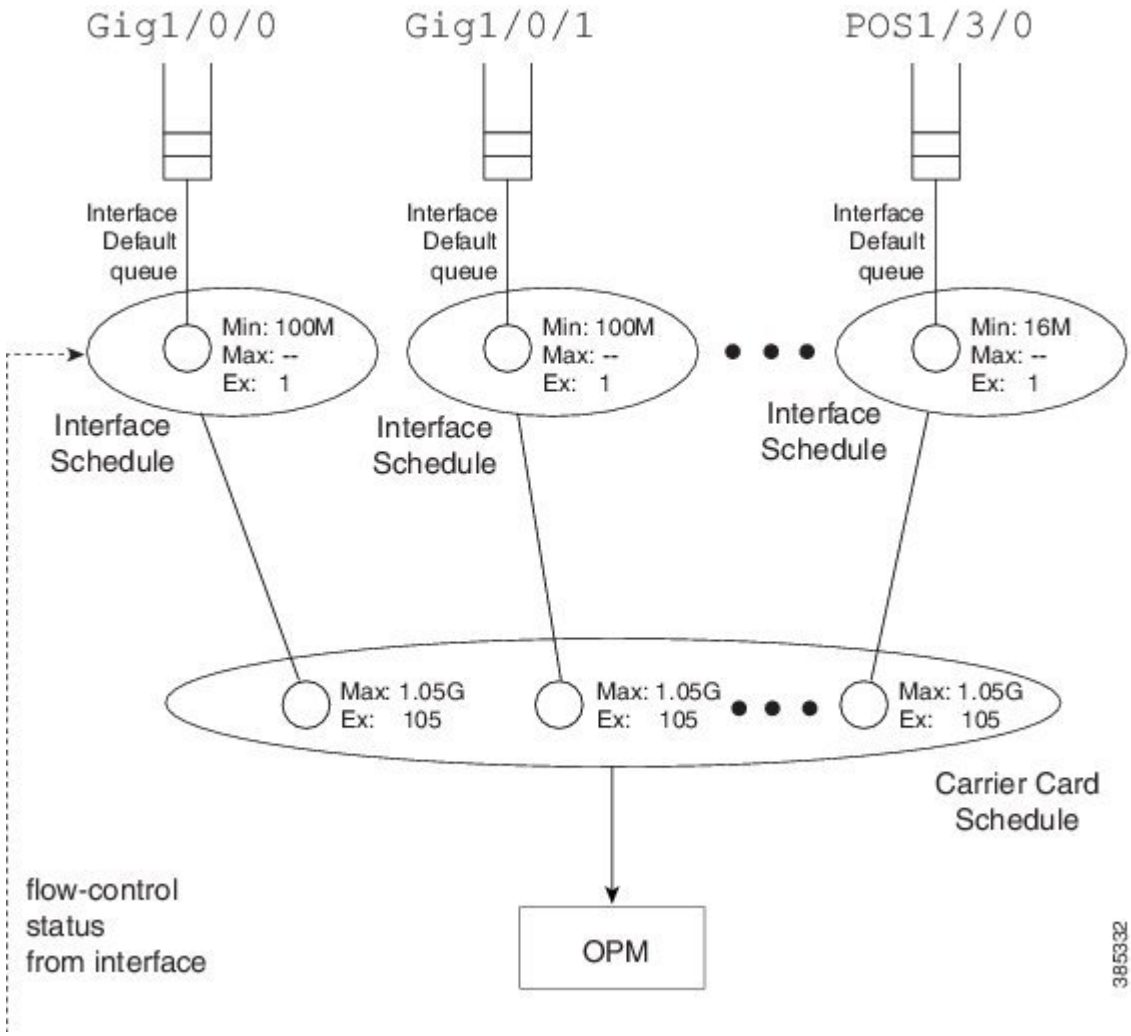
Policy-Maps Attached to Logical Interfaces

Earlier in this chapter, we delineated the two primary methods for creating scheduling hierarchies: QoS policies attached to logical interfaces and hierarchical policy-maps. In prior examples, we outlined polices attached to logical interfaces. Let's explore this scenario in more detail.

Interface Scheduling

Before looking at how a policy on a logical interface alters the hierarchy, we need to carefully examine the interface schedule and hierarchy that exist before any QoS policy is applied:

Figure 46: Interface Schedule and Hierarchy before Application of QoS Policy



The *OPM* (Output Packet Module) sits at the root of the scheduling hierarchy. Upon receiving a packet handle, it fetches the actual packet from memory and pushes it towards the physical interface.

Directly below the OPM layer (from a decision-making perspective) you will find the carrier card schedule. On modular platforms we find one such schedule per slot whereas on fixed systems we have one for the entire system.

Consider a modular chassis with one slot housing an SIP10 that has a 10 Gbps link over the backplane to the ESP (Embedded Services Processor – also termed the *forwarding processor*). The SIP10 can hold 4 SPAs (Shared Port Adapters) where each could have interface(s) totaling at most 10 Gbps capacity. If you combine SPAs in the SIP that exceed the backplane capacity, that link might be a congestion point. Should this occur, the carrier card schedule ensures fairness between interfaces; the excess weight for each interface is proportional to the interface speed.

To *condition traffic* within a platform, we set the Max value in the carrier card schedule for each interface to slightly exceed the interface's bandwidth. We want to send enough traffic towards a physical interface such that we never underrun (starve) that interface. Furthermore, we need to quit sending whenever the interface indicates that its egress buffers are filling, which could happen when an interface receives a pause frame from a downstream device, a serial interface expands its data by bit or byte stuffing, etc.

Here is the key: We push traffic towards a physical interface such that it always has data to send down the wire and we temporarily *pause sending* whenever the interface indicates that it has sufficient data buffered.

An interface directs us to stop sending traffic through a *flow-control message*. By design, a schedule (not a schedule entry) responds to this message - it stops sending. For this reason we must always have an interface schedule for every physical interface in the box. The *interface default queue* (the queue used in absence of QoS) is a child of this interface schedule.

Each interface can send distinct high and low priority flow control messages (to the interface schedule), maintaining distinct buffers and queues for priority and bandwidth traffic:

If the schedule receives a message that bandwidth traffic buffers are filling it will pause such traffic but continue to forward priority traffic.

If we receive a message that priority buffers are filling we will pause sending any packets until the congestion clears.

This scheme extends the concept of priority propagation to the physical interface (recall that this connotes whether a packet handle stems from a priority or bandwidth class) and minimizes jitter to industry leading levels for latency sensitive traffic.

Shape on Parent, or Queue on Child

Now let's look at a typical policy that might be attached to a logical interface (a construct referred to as *shape on parent* or *queue on child*):

```
policy-map child100
  class voice
    priority
    police cir 100m
  class critical-data
    bandwidth 300000
!
policy-map parent100
  class class-default
    shape average 900m
    service-policy child100
!
int g1/0/0.100
  encaps dot1q 100
```

```
service-policy out parent100
```

In this construct, you are required to configure a shaper in the parent policy(`shape average 900M`). The original intent of this construct was to apportion bandwidth to each logical interface. We consider the shape (Max) rate to be the bandwidth owned by that logical interface and allow the child policy to apportion bandwidth within that owned share.

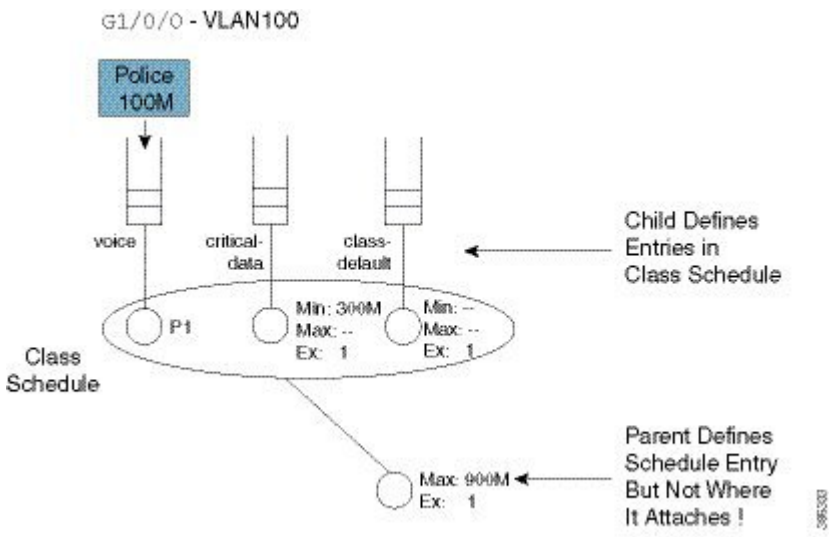
One useful application of this construct is to condition traffic for a remote site. For example, let's say that your corporate hub has a GigabitEthernet link but is sending traffic to a remote branch with a T1 connection. You want to send traffic at the rate the remote branch can receive it. To avoid potentially dropping packets in the provider device that offers service to that branch, you would configure the parent shaper at a T1 rate and queue packets on the hub. This maintains control of what is forwarded initially if that branch link were a congestion point.

Customers have asked to over-provision the shapers on logical interfaces (representing either individual subscribers or remote sites). The assumption is that all logical interfaces would not necessarily be active at all times. As we want to cap the throughput of an individual subscriber, we don’t want to waste bandwidth if an individual logical interface is not consuming its full allocated share.

So, do we oversubscribe? If yes, to provide fairness under congestion thru excess weight values, you should configure a `bandwidth remaining ratio` in the parent. Furthermore, be aware of what service any individual logical interface would receive under congestion.

Returning to the configuration, here is the resultant hierarchy:

Figure 47: Shape on Parent / Queue on Child Construct

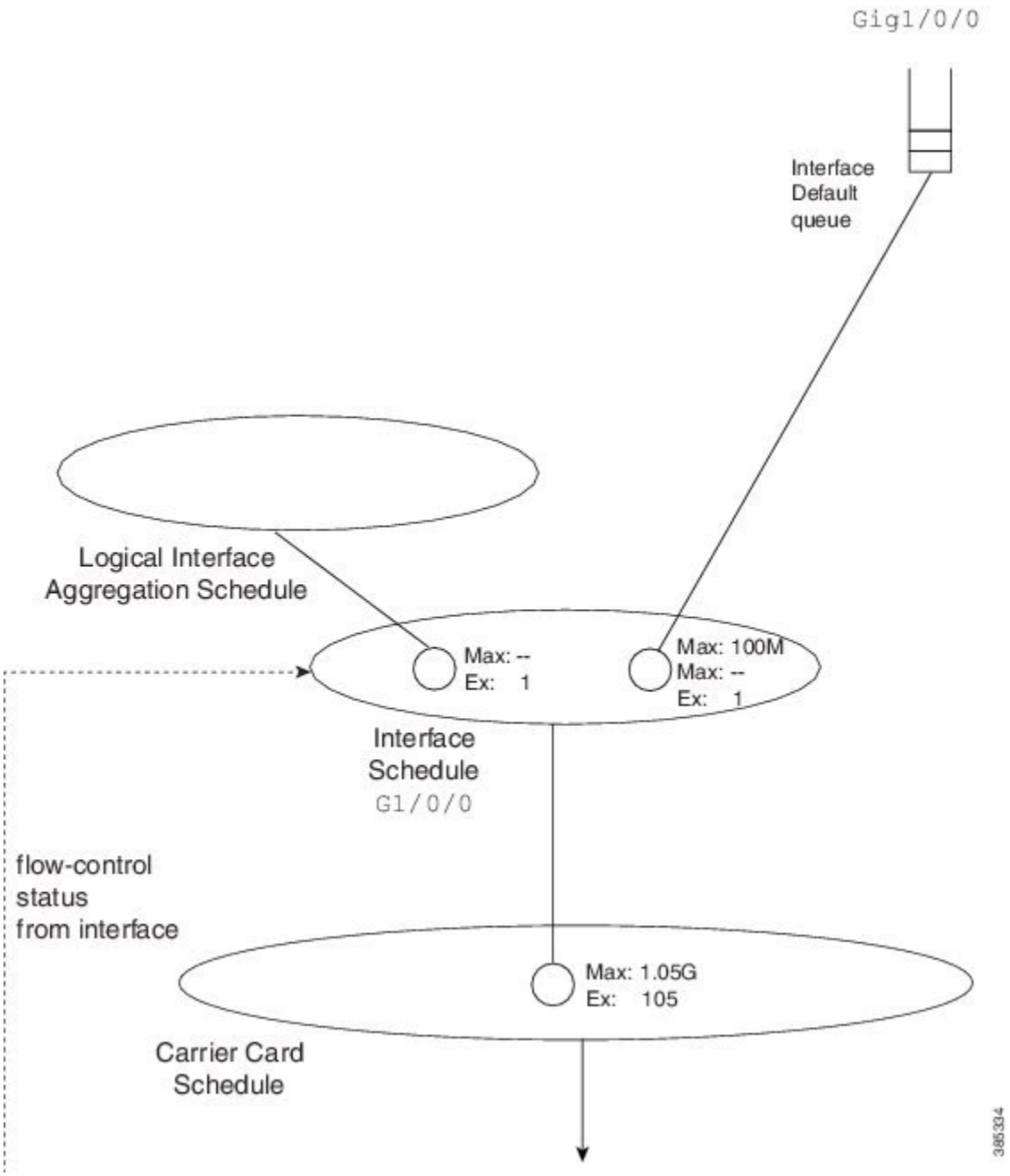


As stated, a child policy defines bandwidth sharing within the logical interface. We usually refer to the queues here (voice, etc.) as *class queues* (with treatment defined by classes within the policy-map) and the schedule at this layer as the *class layer schedule*.

In the parent policy we define a parent shaper (`Max: 900M`) and also the implicit bandwidth share of ‘1’ (`Ex: 1`). Observe that the QoS configuration does not explicitly specify where we should graft this logical interface to the existing interface hierarchy (note the un-attached schedule entry) and the router must know which physical interface a logical interface is associated with to determine where to build the hierarchy.

For a policy on a VLAN, it is evident which interface is involved - we attach the (logical interface) policy in the subinterface configuration. For other interface types (e.g., a tunnel interface), we may need to examine routing information to determine the egress physical interface for that particular logical interface.

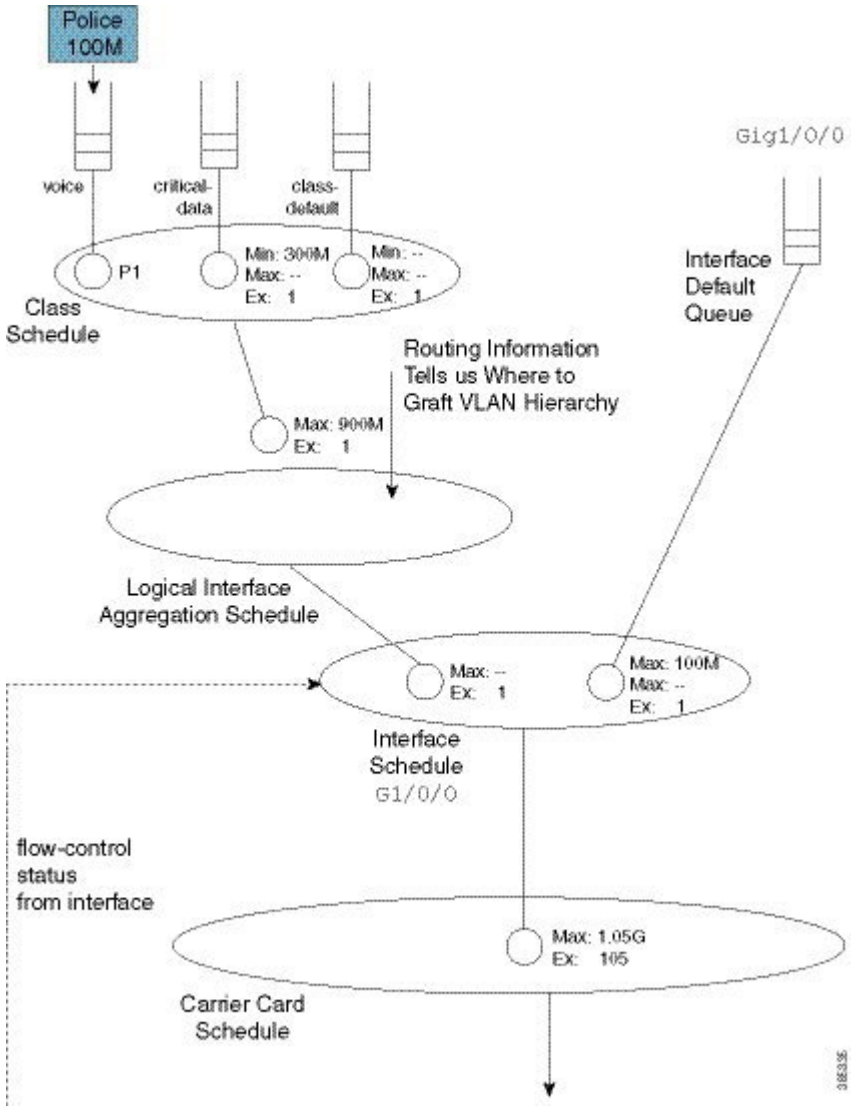
Figure 48: Existing Interface Hierarchy (The World Before the Graft)



After we know which interface is involved, we can modify the hierarchy for that interface. First we create a schedule (the logical interface aggregation) that will serve as a grafting spot for the logical interface hierarchy defined in the shape on parent (or queue on child) policy.

Initially, the interface schedule had a single child, the interface default queue. Now, we create a second child, the *logical interface aggregation schedule*. Observe how the excess weight for this schedule matches that of the interface default queue – it defaults to '1' as always.

Figure 49: Existing Interface Hierarchy (The World After the Graft)



Notice that in the shape on parent policy, we have only class-default with a child policy:

```
policy-map parent100
  class class-default
    shape average 900m
    service-policy child100
```

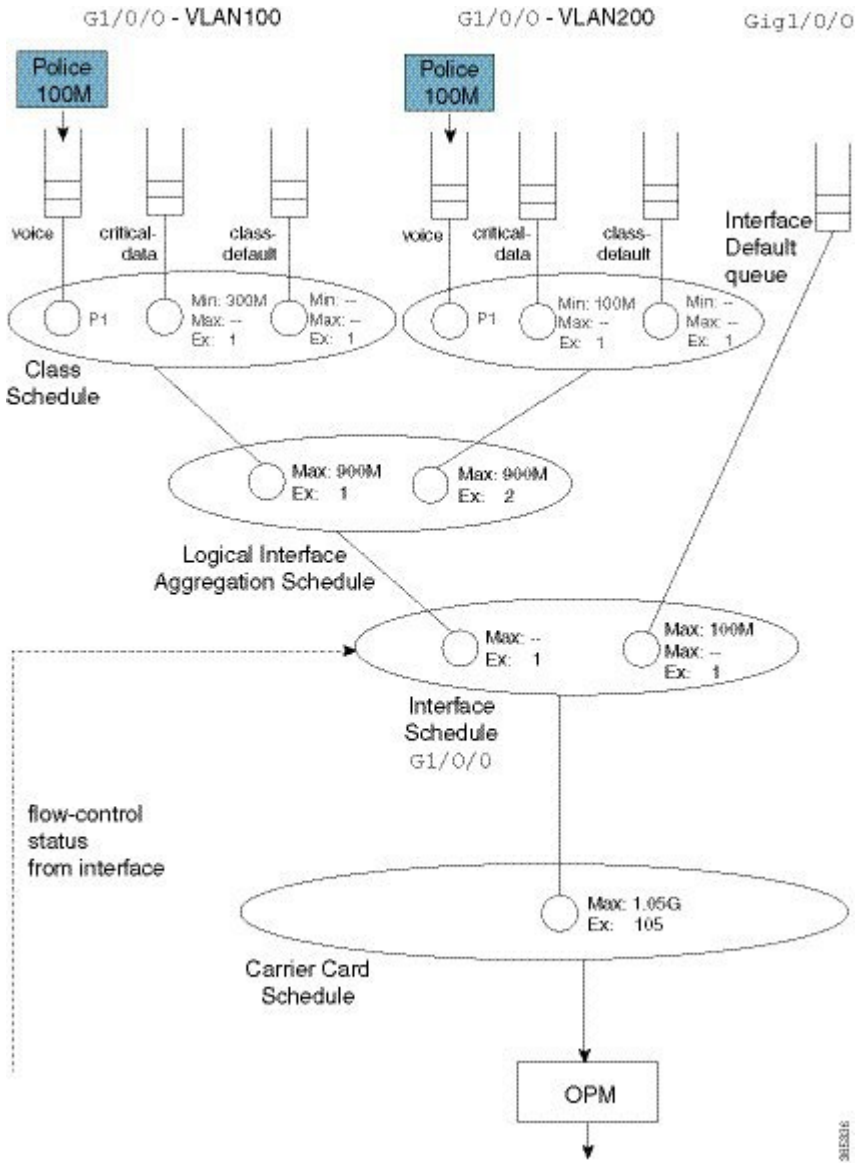
This is a special case where we just define a schedule entry rather than create a schedule for this policy. We refer to this entity as a *collapsed class-default*.

To grasp the significance of this concept, let's add a policy to another VLAN (VLAN200). (Relative to the `policy-map parent100` listed at the beginning of the topic, we have added asterisks):

```
policy-map child200
  class voice
    priority
    police cir 100m
  class critical-data
    bandwidth 100000
!
policy-map parent200
  class class-default
    shape average 900m                    ****
    bandwidth remaining ratio 2
    service-policy child200
!
int g1/0/0.200
  encaps dot1q 200
  service-policy out parent200
```

The complete scheduling hierarchy would now look as follows:

Figure 50: A Complete Hierarchical Scheduling Framework to Handle Congestion and avoid Wasting Bandwidth



Observe that in the second parent policy (the policy to VLAN200) we specified a bandwidth remaining ratio of 2, controlling fairness between VLANS. Recall from the Qos Scheduling chapter the existence of peers in the parent policy of flat policies, which enable us to use either the **bandwidth remaining ratio** or **bandwidth remaining percent** command to specify the excess weight. In the shape on parent policy construct no peers exist. When you configure a QoS policy-map, QoS cannot know what will materialize as peers in the logical interface aggregation schedule. So, neither the **bandwidth remaining ratio** nor the **bandwidth remaining percent** command is supported.

This complete scheduling hierarchy truly highlights the benefits of the Cisco Modular QoS CLI (MQC) and the Hierarchical Scheduling Framework (HQF). For any given interface, the hierarchy is deterministic; we know clearly which packet will be forwarded next. As we have schedules to handle all congestion points, no bandwidth is wasted regardless of where congestion may occur.

Advantages of Policies on Logical Interfaces

The ability to attach policy-maps to logical interfaces offers this significant advantage: management in scaled environments and ease of configuration. For each logical interface, you can reuse or create policy-maps. That is, you might attach a policy-map to each of 1000 VLANs configured on an Ethernet-type interface. To review the QoS statistics for an individual logical interface, you can issue the **show policy-map interface** *interface-name*.

Be aware that the advantages can also be perceived as dangers. If the physical bandwidth available exceeds the sum of your parent shapers, then examining a single logical interface in isolation suffices. However, if the sum of parent shapers exceeds the physical bandwidth available, you need to consider contention between logical interfaces and how much bandwidth an individual interface is truly guaranteed. Viewing an individual interface in isolation may be misleading.

Multiple Policies Definition and Restrictions

We use *Multiple Policies (MPOL)* to describe situations where a policy-map is attached to a logical interface while the policy-map is simultaneously attached to the physical interface to which that logical interface is bound (e.g., a VLAN subinterface and the physical Ethernet interface).

MPOL can also refer to instances where policy-maps are attached to different logical interface types that are bound to the same physical interface. For example, imagine a policy attached to both a VLAN subinterface and a tunnel interface, where both exit the same physical interface.

Currently, the ASR 1000 Series Aggregation Service Router supports a very limited implementation of MPOL. If you have a policy-map attached to a logical interface the only policy you can attach to the physical interface is flat with only class-default and a shaper configured, as in the example below. This topology supports scenarios where the service rate (from a provider) differs from the physical access rate. For example, consider a GigabitEthernet interface connection to your provider where you only pay for 200 Mbps of service. As the service provider will police traffic above that rate, you will want to shape everything (you send) to 200 Mbps and apportion that bandwidth locally.

Note You must attach the policy to the physical interface before you attach it to any logical interface. Furthermore, you can't attach policy-maps to more than one logical interface type bound to a single physical interface.

Returning to the previous example of policies attached to two VLAN subinterfaces (see Shape on Parent, or Queue on Child, on page 107), let's now add a 200 Mbps shaper to the physical interface. The complete configuration would look as follows. The asterisks indicate how this and the previous configuration differ.

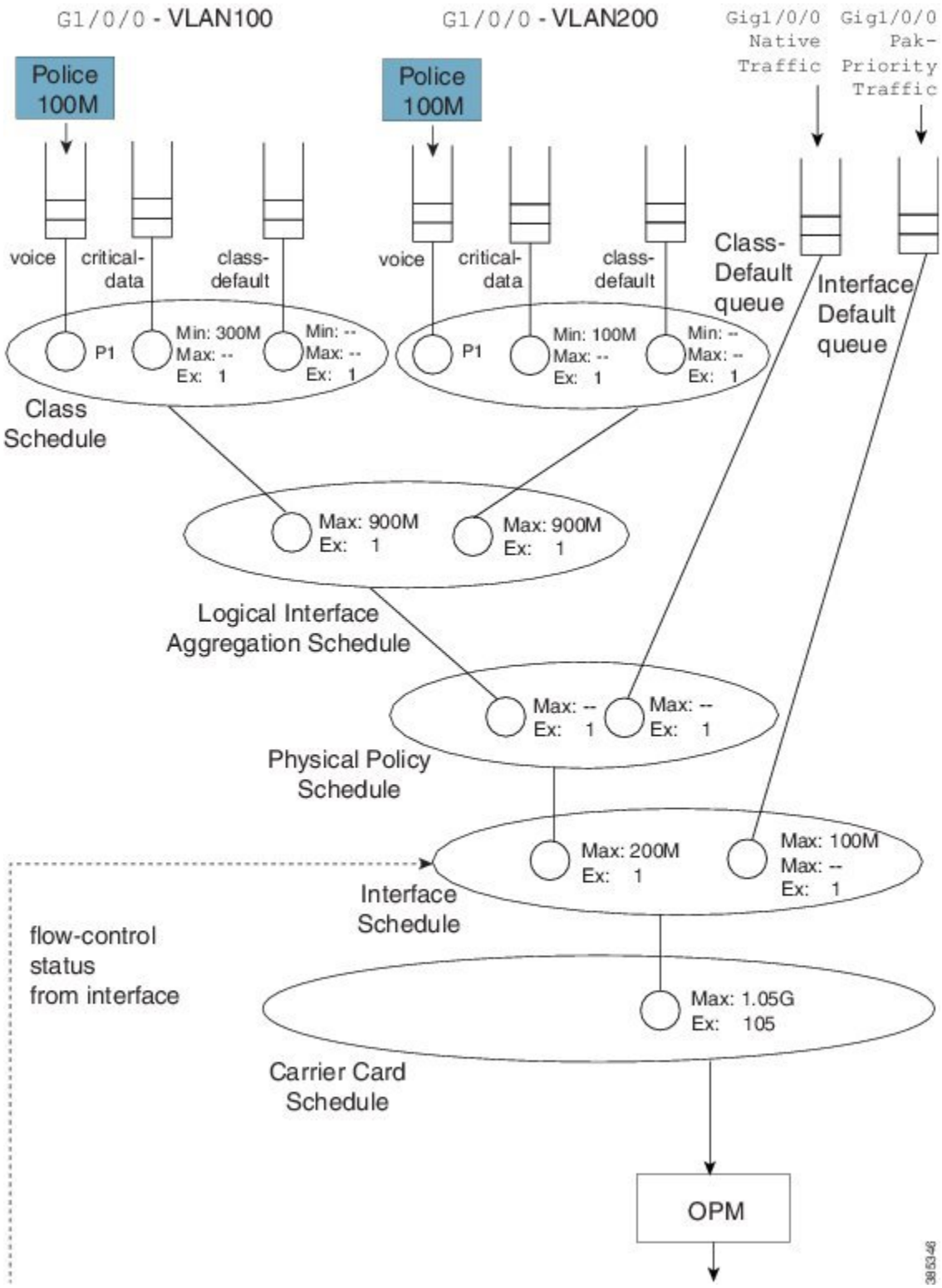
```
policy-map physical-shaper                          ****
  class class-default                               ****
    shape average 200m                              ****
 !
policy-map child100
  class voice
    priority
    police cir 100m
  class critical-data
    bandwidth 300000
!
policy-map child200
  class voice
    priority
```

```
    police cir 100m
  class critical-data
    bandwidth 100000
!
policy-map parent100
  class class-default
    shape average 900m
    service-policy child100
!
policy-map parent200
  class class-default
    shape average 900m
    bandwidth remaining ratio 2
    service-policy child200
!
! Note - must attach physical policy before logical policies
!
int g1/0/0                                          ****
  service-policy output physical-shaper             ****
!
int g1/0/0.100
  encaps dot1q 100
  service-policy out parent100
!
int g1/0/0.200
  encaps dot1q 200
  service-policy out parent200
```

Notice that we have introduced another schedule as well as a queue that will be used for any user traffic sent through the physical interface. The logical interface aggregation schedule has now been created as a child of the physical policy schedule rather than directly as a child of the interface schedule. The combination of traffic through the logical interfaces and user traffic through the physical interface is now shaped to 200 Mbps.

The complete scheduling hierarchy would appear as follows:

Figure 51: Creating a Logical Interface Aggregation as a Child of the Physical Policy Schedule



Hierarchical Policy-Maps

In the previous sections, we showed how hierarchies are constructed when policy-maps are attached to logical interfaces. A second approach is to use *hierarchical policy-maps* and explicitly construct the hierarchy you desire. Using this approach you gain some flexibility but loose some scale. (Recall that with policies on logical interfaces you gained management in scaled environments.) The ASR 1000 Series Aggregation Service Router supports up to 1,000 classes in a policy-map, which means that the largest number of logical interfaces you could represent is 1,000.

To belong to a class within a hierarchical policy-map, a packet must match the child and (any) parent classification rules. In an earlier VLAN example we showed how to use VLAN ID-based classification in a parent class and DSCP-based classification in a child class.

The following configuration shows how we might achieve similar behavior to that with a MPOL-physical shaper (see Multiple Policies Definition and Restrictions, on page 113). Here we use a three-level hierarchical policy-map (the maximum number of layers we support).

The parent policy has only class-default, which means that all traffic through the interface belongs to this class:

```
policy-map physicalshaper
  class class-default
    shape average 200m
    service-policy vlansharing
```

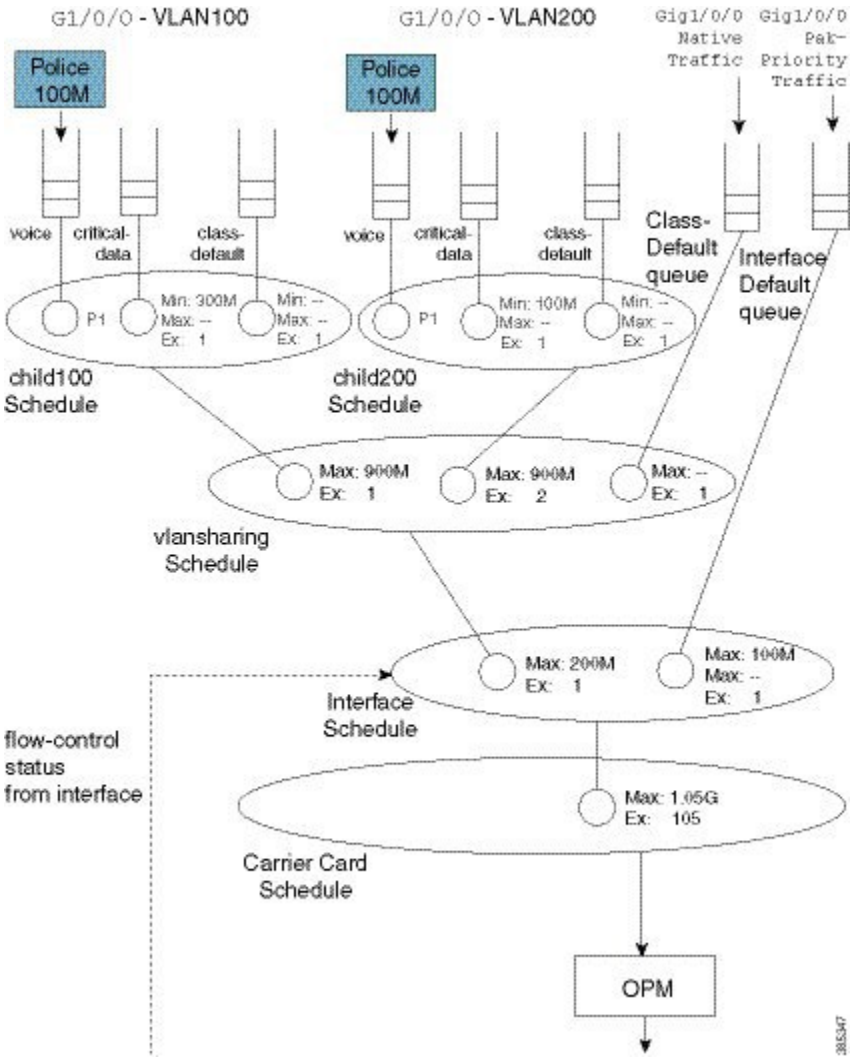
The child level has VLAN-based classification. Traffic belonging to VLAN 100 or VLAN200 will fall into one of the user-defined classes. (Additionally, we have an implicit class-default in this policy that will capture traffic from other VLANs or with no VLAN tag.) Each VLAN class has a policy to further classify traffic based on DSCP:

```
class-map vlan100
  match vlan 100
class-map vlan200
  match vlan 200
class-map voice
  match dscp ef
class-map critical-data
  match dscp af21
        !
policy-map child100
 class voice
   priority
   police cir 100m
 class critical-data
   bandwidth 300000
!
policy-map child200
 class voice
   priority
   police cir 100m
 class critical-data
   bandwidth 100000
!
policy-map vlansharing
 class vlan100
   shape average 900m
   bandwidth remaining ratio 1
   service-policy child100
```

```
 class vlan200
   shape average 900m
   bandwidth remaining ratio 2
   service-policy child200
!
policy-map physicalshaper
   class class-default
     shape average 200m
    service-policy vlansharing
 !
int g1/0/0
  service-policy output physicalshaper
```

A hierarchy constructed based on the above configuration will look as follows:

Figure 52: Hierarchical Policy-Maps to Explicitly-Construct a Hierarchy



If you compare this hierarchy to the previous MPOL example (Figure 25), you will notice some slight differences.

Firstly, native interface traffic (traffic in neither VLAN 100 nor 200) now shares a vlansharing schedule with the schedule entries for each VLAN. In the MPOL example, the native traffic received an equal share to that of all (both) VLANs (1/2 the available bandwidth). In this hierarchy, in contrast, it is guaranteed only 1/(1 + 2 + 1) of available bandwidth as it competes with the VLANs in the same schedule.

Secondly, with a single policy-map on the physical interface you no longer have the ability to look at statistics for a single VLAN only. Compare this code from the MPOL example:

```
int g1/0/0
  service-policy output physical-shaper
!
int g1/0/0.100
  encaps dot1q 100
  service-policy out parent100
!
int g1/0/0.200
  encaps dot1q 200
  service-policy out parent200
```

with this:

```
int g1/0/0
  service-policy output physicalshaper
```

The output of the **show policy-map interface GigabitEthernet1/0/0** command would reflect all levels of the hierarchical policy-map.

Hierarchical policy-maps can add flexibility that is unachievable with policy-maps on logical interfaces. The following examples illustrate this.

Example 1. Add Queues for Different Classes of Traffic

In the discussion of the MPOL example (and captured in the code snippet below), we noted that the physical interface policy could contain only class-default and a shaper in that class:

```
policy-map physical-shaper
  class class-default
    shape average 200m
```

That is, you cannot provide different treatment to unique classes of traffic that were forwarded over the native interface (traffic with no VLAN tag).

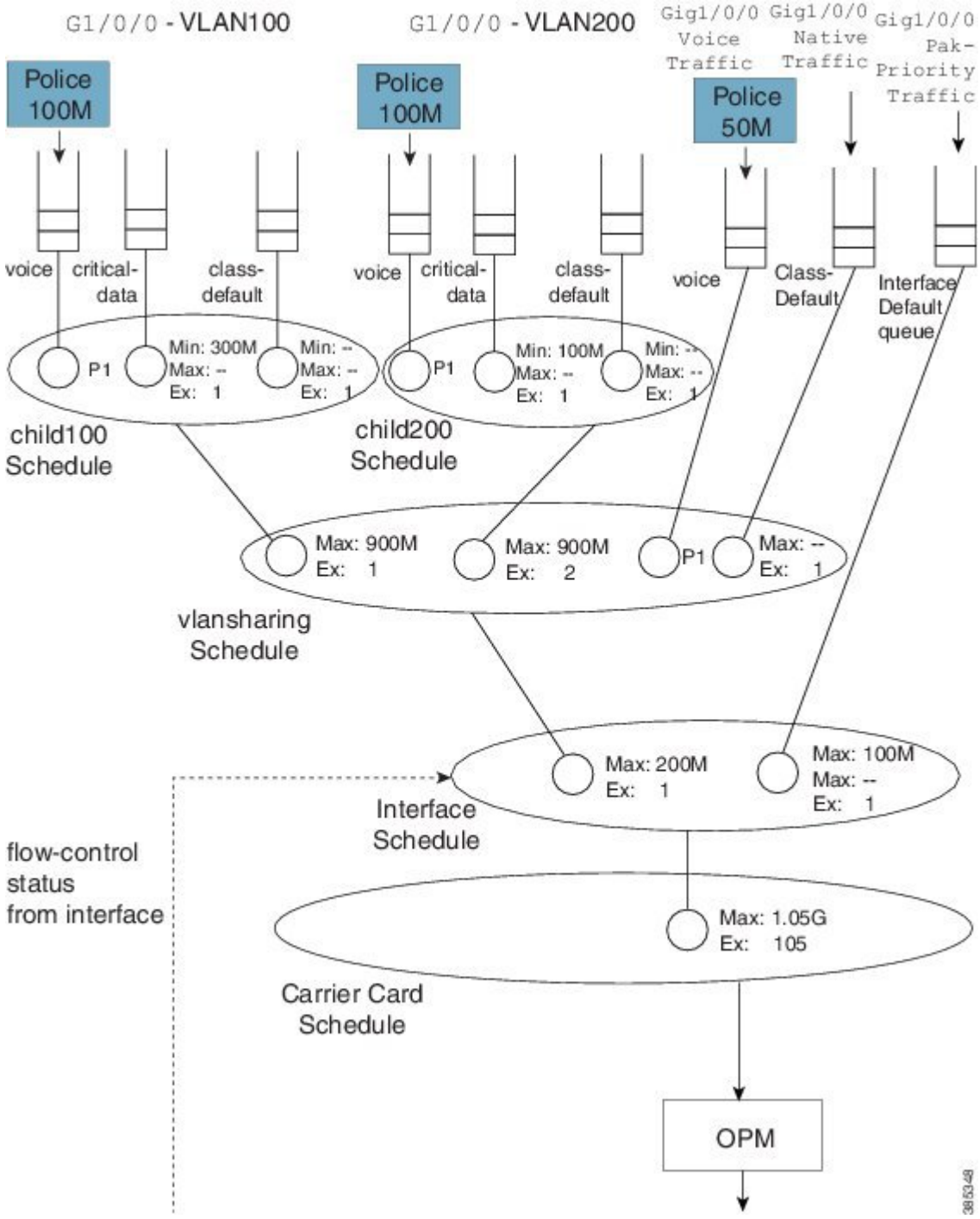
In contrast, with an hierarchical construct, we can add queues for different classes of traffic to forward (over the physical interface). For example, if we wanted to add a priority class for voice traffic over the physical interface, we could modify the vlansharing policy-map as follows (see the asterisks):

```
class-map vlan100
  match vlan 100
class-map vlan200
  match vlan 200
class-map voice
  match dscp ef
class-map critical-data
  match dscp af21
        !
policy-map child100
 class voice
   priority
   police cir 100m
```

```
 class critical-data
   bandwidth 300000
!
policy-map child200
 class voice
   priority
   police cir 100m
 class critical-data
   bandwidth 100000
!
policy-map vlansharing
  class vlan100
    shape average 900m
    bandwidth remaining ratio 1
    service-policy child100
  class vlan200
    shape average 900m
    bandwidth remaining ratio 2
    service-policy child200
  class voice                                ****
    priority                                 ****
    police cir 50m                           ****
!
policy-map physicalshaper
  class class-default
    shape average 200m
    service-policy vlansharing
 !

int g1/0/0
  service-policy output physicalshaper
```

The hierarchy for this configuration would look as follows:

Figure 53: Represent Queues for Different Traffic Classes with a Hierarchical Construct



Notice the new capture that captures any traffic marked with the DSCP codepoint of EF but not tagged with VLAN ID of 100 or 200.

Observe in this hierarchy that P1 traffic from a local queue (Gig1/0/0 Voice Traffic) competes with priority propagation traffic in the VLAN sharing schedule. In such a scenario a local entry configured with priority

is serviced before priority propagation traffic. That is, voice packets from a physical interface (Gig1/0/0) have a slightly higher priority than voice packets from VLAN 100 or 200. To avoid starvation of other classes, we use admission control on the priority queues.

Example 2. Attaching a Policy to Different Logical Interface Types

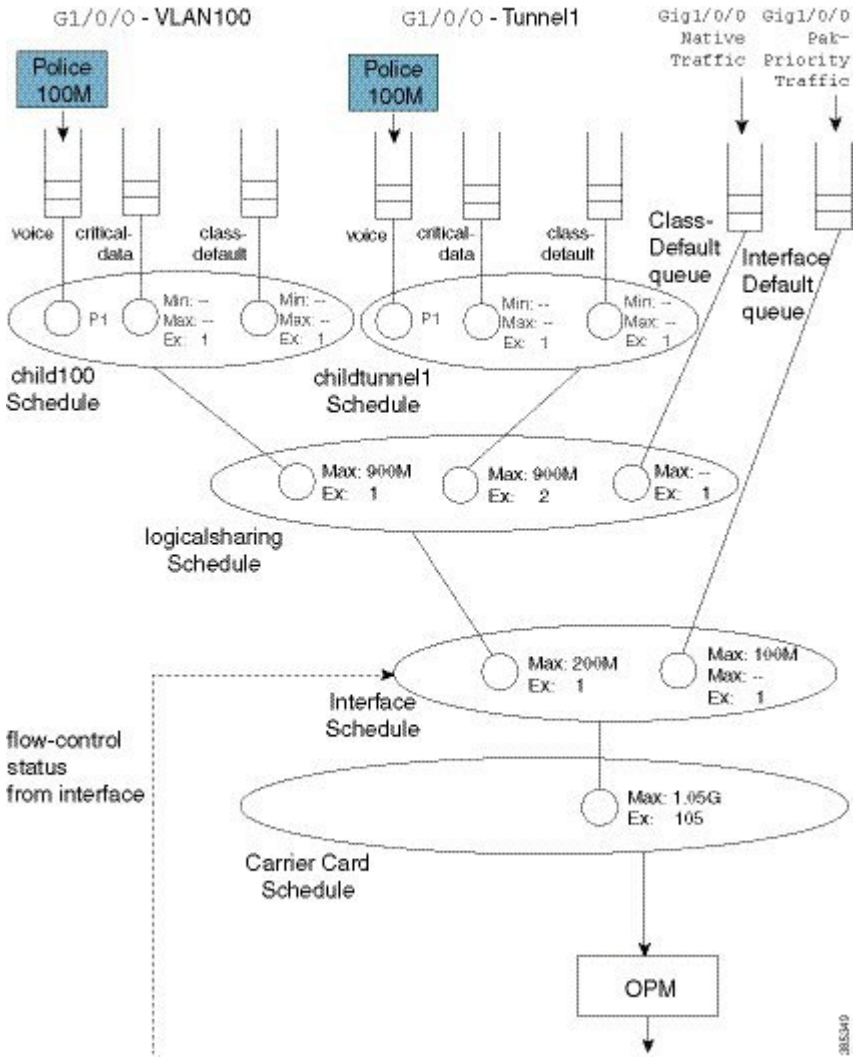
In the section Policy-Maps Attached to Logical Interfaces, on page 106 we indicated that you cannot attach a policy to different logical interface types on the same physical interface. This limitation does not apply to hierarchical class-maps.

Let's say that we want one child schedule for VLAN100 and one child for QoS on a tunnel where both exit the same physical interface. Within the same policy-map, we could classify tunnel traffic using an access list and VLAN traffic using the VLAN ID (see the asterisks):

```
ip access-list extended tunnel1traffic
  permit ip host 192.168.1.1 host 10.0.0.1
!
class-map vlan100
  match vlan 100
class-map tunnel1traffic
  match access-group name tunnel1traffic
!
class-map voice
  match dscp ef
class-map critical-data
  match dscp af21
        !
policy-map child
  class voice
    priority
    police cir 100m
  class critical-data
    bandwidth remaining ratio 1
!
policy-map logicalsharing                  ****
  class vlan100
    shape average 900m
    bandwidth remaining ratio 1
    service-policy child
  class tunnel1traffic
    shape average 900m
    bandwidth remaining ratio 2
    service-policy child
!
policy-map physicalshaper
  class class-default
    shape average 200m
    service-policy vlansharing
 !

int g1/0/0
  service-policy output physicalshaper
```

The hierarchy for this configuration would look as follows:

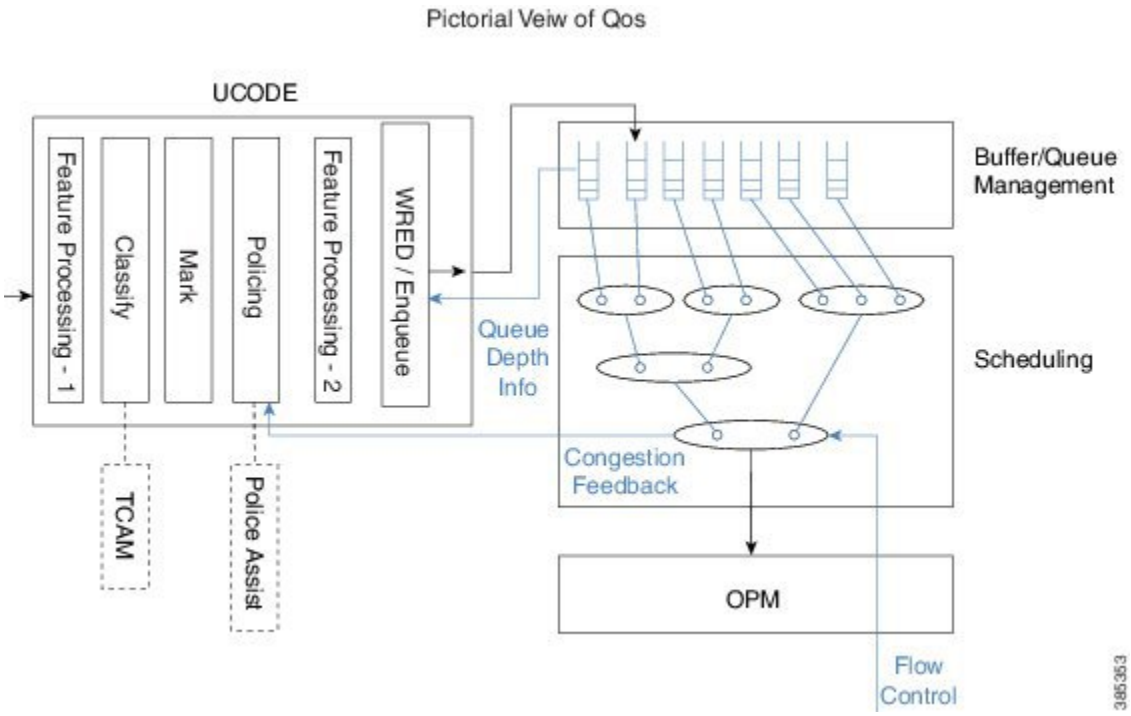
Figure 54: Attaching a Policy to Different Logical Interface Types



A Note on Overhead Accounting

In the policing chapter we introduced the concept of *policing length* (how we perceive a packet's length when a policer evaluates conformance to a configured police rate; see What's Included in the Policer-Rate Calculation (Overhead Accounting), on page 267). Similarly, in the scheduling chapter we introduced the concept of *scheduling length* (how we consider a packet's length when evaluating conformance to a configured scheduler rate; see What's Included in Scheduling Rate Calculations (Overhead Accounting), on page 38.) By convention, in both cases we include the Layer 2 header and datagram lengths and exclude CRC or interpacket overhead.

With an hierarchical scheduling construct, you might encounter instances where the policing and scheduling lengths differ. To understand this let's examine the execution order of features.



On the ASR 1000 Series Aggregation Services Router, queuing and scheduling is performed in hardware. After we enqueue a packet, hardware assumes control and no further processing is performed – the packet must have all headers and be prepared to traverse the wire. As expected, non-queuing features are performed in microcode on one of the processing elements (with hardware assists, in some instances).

Consider two scenarios.

Configuring a QoS-queuing policy on a GRE tunnel

When we classify an incoming IP packet (ultimately encapsulated in an outer IP/GRE header), we examine just the original IP packet. Consequently, classification statistics will exclude the outer IP/GRE headers as they are missing at the time. As the pictorial view indicates, we perform marking and evaluate policers at this time. Similar to the classification length, the policing length will include neither the outer IP/GRE headers nor any egress Layer 2 header, as we don't yet know which physical interface or encapsulation type the packet will egress. After QoS non-queuing features we continue processing the packet by adding the outer IP/GRE header and appropriate Layer 2 header for the final egress interface. When all processing concludes, we pass the packet to the WRED/Enqueue block. This action places the packet on the appropriate egress queue in hardware with all headers added; the scheduling length now includes the outer IP/GRE and Layer 2 headers.

Configuring the QoS policy on the physical egress interface

The results differ. When we examine features on the tunnel no QoS is configured and so we proceed to feature processing. Before reaching the QoS policy, we complete all tunnel processing and add egress headers. So, the classification statistics and policing length will now include the outer headers; policing and scheduling lengths will match.

Verification

In all QoS configuration work, the primary tool to verify hierarchical scheduling configurations is the **show policy-map interface** *interface-name* command. The output of this command is organized hierarchally, reflecting how we stratify the configuration.

For example, with a hierarchical policy attached to a physical interface you could use the **show policy-map interface** *interface-name***| include Class** | to display that hierarchy:

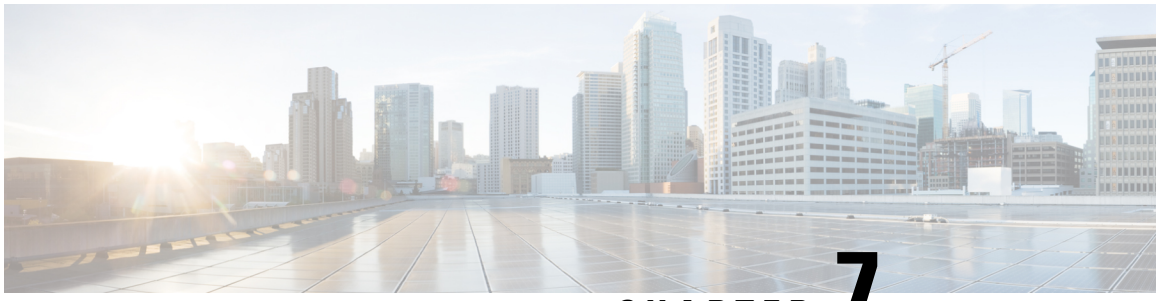
```
show policy-map int g1/0/0 | inc Class

    Class-map: class-default (match-any)
        Class-map: vlan100 (match-all)
            Class-map: voice (match-all)
            Class-map: critical-data (match-all)
            Class-map: class-default (match-any)
        Class-map: vlan200 (match-all)
            Class-map: voice (match-all)
            Class-map: critical-data (match-all)
            Class-map: class-default (match-any)
        Class-map: vlan300 (match-all)
            Class-map: voice (match-all)
            Class-map: class-default (match-any)
        Class-map: voice (match-all)
        Class-map: class-default (match-any)
```

In this example we have attached a 3-level hierarchical policy to interface GigabitEthernet1/0/0. Indentation in the class-map conveys that hierarchy. Within any class that includes a child policy, the `Service-policy: <policy-map name>` indicates that the next-indented section pertains to the child policy:

```
  Class-map: vlan100 (match-all)
          0 packets, 0 bytes
          5 minute offered rate 0000 bps, drop rate 0000 bps
          Match: vlan  100
          Queueing                                                  ****
          queue limit 3748 packets                                  ****
          (queue depth/total drops/no-buffer drops) 0/0/0           ****
          (pkts output/bytes output) 0/0
          shape (average) cir 900000000, bc 3600000, be 3600000
          target shape rate 900000000
          bandwidth remaining ratio 1

          Service-policy : child100

            queue stats for all priority classes:
              Queueing
              queue limit 512 packets
              (queue depth/total drops/no-buffer drops) 0/0/0
              (pkts output/bytes output) 0/0

            Class-map: voice (match-all)
              0 packets, 0 bytes
              5 minute offered rate 0000 bps, drop rate 0000 bps
              Match:  dscp ef (46)
              Priority: Strict, b/w exceed drops: 0

              police:
                  cir 100000000 bps, bc 3125000 bytes
                conformed 0 packets, 0 bytes; actions:
                  transmit
```

```
    exceeded 0 packets, 0 bytes; actions:
      drop
    conformed 0000 bps, exceeded 0000 bps

Class-map: critical-data (match-all)
  0 packets, 0 bytes
  5 minute offered rate 0000 bps, drop rate 0000 bps
  Match:  dscp af11 (10)
  Match:  dscp af21 (18)
  Queueing
  queue limit 1249 packets
  (queue depth/total drops/no-buffer drops) 0/0/0
  (pkts output/bytes output) 0/0
  bandwidth 300000 kbps

Class-map: class-default (match-any)
  0 packets, 0 bytes
  5 minute offered rate 0000 bps, drop rate 0000 bps
  Match: any

  queue limit 3748 packets
  (queue depth/total drops/no-buffer drops) 0/0/0
  (pkts output/bytes output) 0/0
```

Regarding **show** command output for a policy containing hierarchical scheduling, observe that any queue-related information in the parent class is meaningless (highlighted by asterisks in the example above). The output format for the **show policy-map interface** command was created at a time when IOS truly implemented a hierarchy of queues in software. The ASR 1000 Series Aggregation Services Router hardware implements a hierarchy of schedules and queues, which only exist at the leaf. Although the IOS control plane still calculates and displays a queue-limit, it never uses it. So tuning this value is fruitless.



CHAPTER 7

Legacy QoS Command Deprecation

The functionality provided by these hidden commands has been replaced by similar functionality provided via the modular QoS CLI (MQC). The MQC is a set of a platform-independent commands for configuring QoS on Cisco platforms. This means that you must now provision QoS by defining traffic classes, creating traffic policies containing those classes, and attaching those policies to the desired interfaces. This document lists the hidden commands and their replacement MQC commands.

Finding Feature Information

Your software release may not support all the features documented in this module. For the latest caveats and feature information, see Bug Search Tool and the release notes for your platform and software release. To find information about the features documented in this module, and to see a list of the releases in which each feature is supported, see the feature information table.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to https://cfnng.cisco.com/. An account on Cisco.com is not required.

Information About Legacy QoS Command Deprecation

QoS Features Applied Using the MQC

The MQC structure lets you define a traffic class (also called a class map), create a traffic policy (also called a policy-map), and attach the traffic policy to an interface. This comprises the following three high-level steps.

1. Define a traffic class by using the **class-map** command. A traffic class is used to classify traffic.
2. Create a traffic policy by using the **policy-map** command. A traffic policy contains a traffic class and one or more QoS features that will be applied to the traffic class. The QoS features in the traffic policy determine how to treat the classified traffic.
3. Attach the traffic policy to the interface by using the **service-policy** command.

Steps 1 and 3 do not involve legacy QoS hidden commands, which means that they are not within the scope of this document. For more information about these two steps, see the " Applying QoS Features Using the MQC " module in the *Quality of Service Solutions Configuration Guide* .

Legacy Commands Being Hidden

The table below lists the commands that have been hidden or removed. The table also lists their replacement commands (or sequence of commands).

Table 3: Map of Hidden, Removed or Unsupported Commands to Their Replacement Commands

Hidden, Removed or Unsupported Commands	Replacement MQC Command Sequence
Configuring Weighted Random Early Detection or Distributed Weighted Random Early Detection Parameter Groups	

Commands
• random-detect-group
• random-detect (per VC)
Note This command is not supported in Cisco IOS Release 15.0(1)S.
Command Usage
`Router(config)# random-detect-group group-name [dscp-based|prec-based]`
`Router(config)# interface atm type number`
`Router(config-if)# pvc [name] vpi/vci`
`Router(config-if-atm-vc)# random-detect [attach group-name ]`
	Command Usage
	None (this functionality no longer exists).

Configuring Weighted Random Early Detection	

Hidden, Removed or Unsupported Commands	Replacement MQC Command Sequence

Commands
• random-detect
• random-detect dscp
• random-detect (dscp-based keyword)
• random-detect flow
• random-detect exponential-weighting-constant
• random-detect (prec-based keyword)
• random-detect precedence
Command Usage
`Router(config)# interface type number`
`Router(config-if)# random-detect [number]`
`Router(config-if)# random-detect exponential-weighting-constant exponent`
`Router(config-if)# random-detect flow`
`Router(config-if)# random-detect precedence {precedence|rsvp} min-threshold max-threshold max-probability-denominator`
`Router(config-if)# random-detect prec-based`
`Router(config-if)# random-detect dscp-based`
`Router(config-if)# random-detect dscp dscp-value min-threshold max-threshold[max-probability-denominator]`
	Command Usage
	`Router(config)# policy-map`
	`policy-map-name`
	`Router(config-pmap)# class class-default`
	`Router(config-pmap-c)# random-detect dscp dscp-value min-threshold max-threshold[ mark-probability-denominator]`
	`Router(config-pmap-c)# random-detect clp clp-value min-threshold max-threshold[mark-probability-denominator]`
	`Router(config-pmap-c)# random-detect cos cos-value min-threshold max-threshold[mark-probability-denominator]`
	`Router(config-pmap-c)# random-detect discard-class discard-class-value min-threshold max-threshold[ mark-probability-denominator]`
	`Router(config-pmap-c)# random-detectprecedence ip-precedence min-threshold max-threshold[mark-probability-denominator]`
	`Router(config-pmap-c)# random-detect precedence-based`
	`Router(config-pmap-c)# random-detect ecn`
	`Router(config-pmap-c)# random-detect exponential-weighting-constant exponent`
	`Router(config-pmap-c)# random-detect cos-based`
	`Router(config-pmap-c)# random-detect dscp-based`

Commands
• random-detect flow
• random-detect flow average-depth-factor
• random-detect flow count
Command Usage
`Router(config)# interface type number`
`Router(config-if)# random-detect [number]`
`Router(config-if)# random-detect flow`
`Router(config-if)# random-detect flow count number`
`Router(config-if)# random-detect flow average-depth-factor scaling-factor`
	Command Usage
	None (this functionality no longer exists).

Configuring Bandwidth Allocation	

Hidden, Removed or Unsupported Commands	Replacement MQC Command Sequence

Commands
• max-reserved-bandwidth
Command Usage
`Router(config)# interface type number`
`Router(config-if)# max-reserved-bandwidth percentage`
	Command Usage
	`Router(config)# policy-map policy-map-name`
	`Router(config-pmap)# class class-default`
	`Router(config-pmap-c)# bandwidth{bandwidth-in-kbps |remaining percent percentage | percent percentage}`

Configuring Custom Queueing	

Commands
• custom-queue-list
Note This command is not supported in Cisco IOS Release 15.0(1)S.
Command Usage
`Router(config)# interface type number`
`Router(config-if)#`
`custom-queue-list[list-number]`
	Command Usage
	`Router(config)# policy-map policy-map-name`
	`Router(config-pmap)# class class-default`
	`Router(config-pmap-c)# bandwidth{ bandwidth-in-kbps |remaining percent percentage |percent percentage}`

Configuring Priority Queueing	

Commands
• ip rtp priority
• ip rtp reserve
Command Usage
`Router(config)# interface type number`
`Router(config-if)# ip rtp priority starting-port-number port-range bandwidth`
`Router(config)# interface type number`
`Router(config-if)# ip rtp reserve lowest-udp-port range-of-ports [maximum-bandwidth] 1000`
	Command Usage
	`Router(config)# policy-map policy-map-name`
	`Router(config-pmap)# class class-name`
	`Router(config-pmap-c)# priority`

Configuring Weighted Fair Queueing	

Hidden, Removed or Unsupported Commands	Replacement MQC Command Sequence

Commands
• fair-queue (WFQ)
Command Usage (Cisco IOS Release 15.0(1)S)
`Router(config)# interface type number`
`Router(config-if)# fair-queue`
Command Usage (Cisco IOS Release 15.1(3)T)
`Router(config)# interfacetype number`
`Router(config-if)# fair-queue [congestive-discard-threshold [dynamic-queue-count [reserved-queue-count]]]`
	Command Usage (Cisco IOS Release 15.0(1)S)
	`Router(config)# policy-map policy-map-name`
	`Router(config-pmap)# class class-default`
	`Router(config-pmap-c)# fair-queue`
	Command Usage (Cisco IOS Release 15.1(3)T)
	`Router(config)# policy-map policy-map-name`
	`Router(config-pmap)# class class-default`
	`Router(config-pmap-c)# fair-queue[dynamic-queues ]`

Assigning a Priority Group to an Interface	

Commands
• priority-group
Note This command is not supported in Cisco IOS Release 15.0(1)S.
Command Usage
`Router(config)# interface type number`
`Router(config-if)# priority-group list-number`
	Command Usage
	`Router(config)# policy-map policy-map-name`
	`Router(config-pmap)# class class-default`
	`Router(config-pmap-c)# priority`
	`Router(config-pmap-c)# priority bandwidth-in-kbps [burst-in-bytes]`
	`Router(config-pmap-c)# priority percent percent [burst-in-bytes]`
	`Router(config-pmap-c)# priority level level`
	`Router(config-pmap-c)# priority level level [bandwidth-in-kbps [burst-in-bytes]]`
	`Router(config-pmap-c)# priority level level[percent percent [burst-in-bytes]]`

Configuring the Threshold for Discarding DE Packets from a Switched PVC Traffic Shaping Queue	

Commands
• frame-relay congestion threshold de
Command Usage
`Router(config)# map-class frame-relay map-class-name`
`Router(config-map-class)# frame-relay congestion threshold de percentage`
	Command Usage
	`Router(config)# policy-map policy-map-name1`
	`Router(config-pmap)# class class-default`
	`Router(config-pmap-c)# random-detect discard-class-based`
	`Router(config-pmap-c)# random-detect discard-class discard-class min-threshold max-threshold`
	`Router(config-pmap-c)# exit`
	`Router(config-pmap)# exit`
	`Router(config)# policy-map shape`
	`Router(config-pmap)# class class-default`
	`Router(config-pmap-c)# shape average rate`
	`Router(config-pmap-c)# service-policy policy-map-name1`
	`Router(config-pmap-c)# exit`
	`Router(config-pmap)# exit`
	`Router(config)# policy-map policy-map-name2`
	`Router(config-pmap)# class class-name`
	`Router(config-pmap-c)# set discard-classdiscard-class`

Hidden, Removed or Unsupported Commands	Replacement MQC Command Sequence
Configuring Frame Relay Custom Queueing for Virtual Circuits	

Commands
• frame-relay custom-queue-list
Command Usage
`Router(config)# map-class frame-relay map-class-name`
`Router(config-map-class)# frame-relay custom-queue-list list-number`
	Command Usage
	`Router(config)# policy-map policy-map-name`
	`Router(config-pmap)# class class-default`
	`Router(config-pmap-c)# bandwidth{`*bandwidth-in-kbps* `| remaining percent` *percentage* `| percent`*percentage*`}`

Configuring Frame Relay ECN Bits Threshold	

Commands
• frame-relay congestion threshold ecn
Command Usage
`Router(config)# map-class frame-relay map-class-name`
`Router(config-map-class)# frame-relay congestion threshold ecn percentage`
	Command Usage
	None (this functionality no longer exists).
	The closest equivalent is MQC traffic shaping (not based on ECN).
	`Router(config)# policy-map policy-map-name`
	`Router(config-pmap)# class class-default`
	`Router(config-pmap-c)# shape average rate`

Configuring Frame Relay Weighted Fair Queueing	

Commands
• frame-relay fair-queue
Command Usage
`Router(config)# map-class frame-relay map-class-name`
`Router(config-map-class)# frame-relay fair-queue [discard-threshold [dynamic-queue-count[reserved-queue-count [buffer-limit]]]]`
	Command Usage
	`Router(config)# policy-map policy-map-name`
	`Router(config-pmap)# class class-default`
	`Router(config-pmap-c)# fair-queue`
	`Router(config-pmap-c)# fair-queue queue-limit packets`
	Note The **queue-limit** *packets* keyword and argument pair is not supported in Cisco IOS Release 15.1(3)T.

Configuring Frame Relay Priority Queueing on a PVC	

Commands
• frame-relay ip rtp priority
Command Usage
`Router(config)# map-class frame-relay map-class-name`
`Router(config-map-class)# frame-relay ip rtp priority starting-port-number port-range bandwidth`
	Command Usage
	`Router(config)# policy-map policy-map-name`
	`Router(config-pmap)# class class-name`
	`Router(config-pmap-c)# priority bandwidth-in-kbps [burst-in-bytes]`

Hidden, Removed or Unsupported Commands	Replacement MQC Command Sequence
Assigning a Priority Queue to Virtual Circuits Associated with a Map Class	

Commands
• frame-relay priority-group
Command Usage
`Router(config)# map-class frame-relay map-class-name`
`Router(config-map-class)# frame-relay priority-group group-number`
	Command Usage
	`Router(config)# policy-map policy-map-name`
	`Router(config-pmap)# class class-default`
	`Router(config-pmap-c)# priority`
	`Router(config-pmap-c)# priority bandwidth-in-kbps [burst-in-bytes]`
	`Router(config-pmap-c)# priority percent percentage [burst-in-bytes]`
	`Router(config-pmap-c)# priority level level [percent percentage [burst-in-bytes]]`
	Note The **priority level**command is not supported in Cisco IOS Release 15.1(3)T.

Configuring the Frame Relay Rate Adjustment to BECN	

Commands
• frame-relay adaptive-shaping (becn keyword)
Command Usage
`Router(config)# map-class frame-relay map-class-name`
`Router(config-map-class)# frame-relay adaptive-shaping becn`
	Command Usage
	None (this functionality no longer exists). The closest equivalent is MQC traffic shaping (not based on BECN).
	`Router(config)# policy-map policy-map-name`
	`Router(config-pmap)# class class-default`
	`Router(config-pmap-c)# shape adaptive rate`

Configuring the Frame Relay Rate Adjustment to ForeSight Messages	

Commands
• frame-relay adaptive-shaping (foresight keyword)
Command Usage
`Router(config)# map-class frame-relay map-class-name`
`Router(config)# frame-relay adaptive-shaping foresight`
	Command Usage
	None (this functionality no longer exists).

Enabling Frame Relay Traffic-Shaping FECNs as BECNs	

Hidden, Removed or Unsupported Commands	Replacement MQC Command Sequence

Commands
• frame-relay fecn-adapt
Command Usage
`Router(config)# map-class frame-relay map-class-name`
`Router(config-map-class)#frame-relay fecn-adapt`
	Command Usage
	None (this functionality no longer exists). The closest equivalent is MQC traffic shaping (not based on FECN/BECN).
	`Router(config)# policy-map policy-map-name`
	`Router(config-pmap)# class class-default`
	`Router(config-pmap-c)# shape average rate`

Configuring the Frame Relay Enhanced Local Management Interface	

Commands
• frame-relay qos-autosense
Note This command has not been hidden in Cisco IOS Release 15.0(1)S.
Command Usage
`Router(config)# interface type number`
`Router(config-if)#no ip address`
`Router(config-if)# encapsulation frame-relay`
`Router(config-if)# frame-relay lmi-typeansi`
`Router(config-if)# frame-relay traffic-shaping`
`Router(config-if)# frame-relay qos-autosense`
	Command Usage
	None (this functionality no longer exists).

Configuring Frame Relay Minimum Committed Information Rate (MINCIR)	
Commands • frame-relay mincir **Command Usage** `Router(config)# frame-relay mincir {in | out} bps`	**Command Usage** None (this functionality no longer exists).
Configuring Frame Relay Priority to a permanent virtual circuit (PVC)	

Hidden, Removed or Unsupported Commands	Replacement MQC Command Sequence

Commands
• frame-relay interface-queue
Command Usage
`Router(config)# interface type number`
`Router(config-if)#no ip address`
`Router(config-if)# frame-relay interface-queue priority 10 20 30 40`
	Command Usage
	`Router(config)# policy-map policy-map-name`
	`Router(config-pmap)# class class-default`
	`Router(config-pmap-c)# priority`
	`Router(config-pmap)# class class-default`
	`Router(config-pmap-c)# priority`

Configuring Frame Relay Traffic Shaping	

Commands
• frame-relay bc
• frame-relay be
• frame-relay cir
Note In Cisco IOS Release 15.1(3)T, these commands are not hidden, but they are valid only for SVCs (not PVCs).
Command Usage
`Router(config)# map-class frame-relay map-class-name`
`Router(config-map-class)# frame-relay bc {in | out} committed-burst-size-in-bits`
`Router(config-map-class)# frame-relay be {in | out} excess-burst-size-in-bits`
`Router(config-map-class)# frame-relay cir {in | out} bits-per-second`
	Command Usage
	`Router(config)# policy-map policy-map-name`
	`Router(config-pmap)# class class-default`
	`Router(config-pmap-c)# shape average rate`

Configuring Frame Relay Traffic Shaping on a VC	

Commands
• frame-relay traffic-rate
Command Usage
`Router(config)# map-class frame-relay map-class-name`
`Router(config-map-class)# traffic-rate average [peak]`
	Command Usage
	`Router(config)# policy-map policy-map-name`
	`Router(config-pmap)# class class-default`
	`Router(config-pmap-c)# shape average rate`
	`Router(config-pmap-c)# service-policy output traffic-rate` service-policy output traffic-rate

Displaying the Contents of Packets Inside a Queue for an Interface or VC	

Hidden, Removed or Unsupported Commands	Replacement MQC Command Sequence
Commands • show queue **Command Usage** `Router# show queue interface`	**Command Usage** `Router# show policy-map interface`
Displaying Queueing Strategies	
Commands • show queueing **Command Usage** `Router# show queueing`	**Command Usage** `Router# show policy-map interface`
Displaying Weighted Random Early Detection (WRED) Information	
Commands • show interfaces random-detect **Command Usage** `Router# show interfaces [type number] random-detect`	**Command Usage** `Router# show policy-map interface`
Displaying WRED Parameter Groups	
Commands • show random-detect-group **Command Usage** `Router# show random-detect-group`	**Command Usage** `Router# show policy-map interface`
Displaying the Traffic-Shaping Configuration, Queueing, and Statistics	

Hidden, Removed or Unsupported Commands	Replacement MQC Command Sequence

Commands
• show traffic-shape
• show traffic-shape queue
• show traffic-shape statistics
Command Usage
`Router# show traffic-shape [interface-type interface-number]`
`Router# show traffic-shape queue [interface-number [dlci dlci-number]]`
`Router# show traffic-shape statistics [interface-type interface-number]`
	Command Usage
	`Router# show policy-map interface`

Displaying Weighted Fair Queueing Information	
Commands • show interfaces fair-queue **Command Usage** `Router# show interfaces [interface-type interface-number] fair-queue`	**Command Usage** `Router# show policy-map interface`

Additional References

Related Documents

Related Topic	Document Title
Cisco IOS commands	Cisco IOS Master Commands List, All Releases
Defining traffic classes; attaching traffic policies to interfaces	" Applying QoS Features Using the MQC " module in the *Quality of Service Solutions Configuration Guide*
Reference pages for QoS commands	*Cisco IOS Quality of Service Solutions Command Reference*
Reference pages for wide-area networking commands	*Cisco IOS Wide-Area Networking Command Reference*

Technical Assistance

Description	Link
The Cisco Support and Documentation website provides online resources to download documentation, software, and tools. Use these resources to install and configure the software and to troubleshoot and resolve technical issues with Cisco products and technologies. Access to most tools on the Cisco Support and Documentation website requires a Cisco.com user ID and password.	http://www.cisco.com/cisco/web/support/index.html

Feature Information for Legacy QoS Command Deprecation

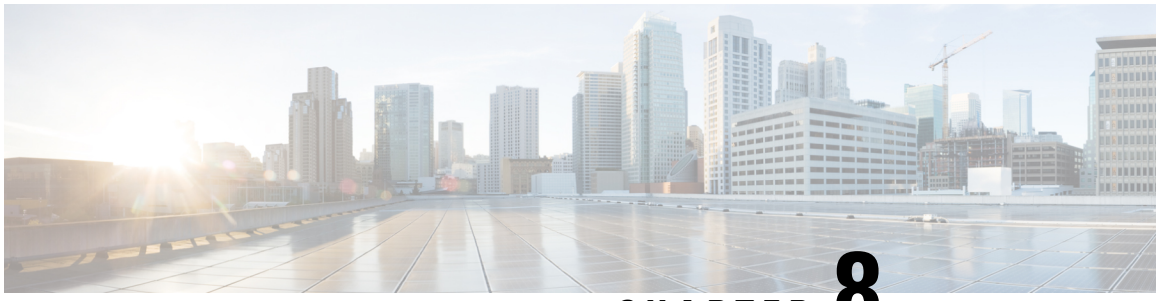
The following table provides release information about the feature or features described in this module. This table lists only the software release that introduced support for a given feature in a given software release train. Unless noted otherwise, subsequent releases of that software release train also support that feature.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to www.cisco.com/go/cfn. An account on Cisco.com is not required.

Table 4: Feature Information for Legacy QoS Command Deprecation

Feature Name	Releases	Feature Information
Legacy QoS Command Deprecation: Hidden Commands	15.0(1)S 15.1(3)T	To streamline Cisco IOS QoS, certain commands have been hidden, which means that if you try to view a hidden command by entering a question mark (**?**) at the command line, the command does not appear. However, if you know the command syntax, you can enter it. These commands will be removed in a future release. The functionality provided by these hidden commands is replaced by similar functionality from the modular QoS CLI (MQC), which is a set of a platform-independent commands for configuring QoS. The following commands were modified: **custom-queue-list**, **fair-queue (WFQ)**, **frame-relay adaptive-shaping** (**becn** keyword), **frame-relay adaptive-shaping** (**foresight** keyword), **frame-relay bc**, **frame-relay be**, **frame-relay cir**, **frame-relay congestion threshold de**, **frame-relay congestion threshold ecn**, **frame-relay custom-queue-list**, **frame-relay fair-queue**, **frame-relay fecn-adapt**, **frame-relay ip rtp priority**, **frame-relay priority-group**, **frame-relay qos-autosense**, **ip rtp priority**, **max-reserved-bandwidth**, **priority-group**, **random-detect**, **random-detect dscp**, **random-detect**(**dscp-based** keyword), **random-detect exponential-weighting-constant**, **random-detect flow**, **random-detect flow average-depth-factor**, **random-detect flow count**, **random-detect**(**prec-based** keyword), **random-detect precedence**, **random-detect-group**, **show interfaces fair-queue**, **show interfaces random-detect**, **show queue**, **show queueing**, **show random-detect-group**, **show traffic-shape**, **show traffic-shape queue**, **show traffic-shape statistics**.

Feature Name	Releases	Feature Information
Legacy QoS Command Deprecation: Hidden Commands	Cisco IOS XE Release 2.6	To streamline Cisco IOS XE QoS, certain commands have been hidden, which means that if you try to view a hidden command by entering a question mark (**?**) at the command line, the command does not appear. However, if you know the command syntax, you can enter it. These commands will be removed in a future release. The functionality provided by these hidden commands is replaced by similar functionality from the modular QoS CLI (MQC), which is a set of a platform-independent commands for configuring QoS. The following commands were modified: **custom-queue-list**, **fair-queue (WFQ)**, **frame-relay adaptive-shaping** (**becn** keyword), **frame-relay adaptive-shaping** (**foresight** keyword), **frame-relay bc**, **frame-relay be**, **frame-relay cir**, **frame-relay congestion threshold de**, **frame-relay congestion threshold ecn**, **frame-relay custom-queue-list**, **frame-relay fair-queue**, **frame-relay fecn-adapt**, **frame-relay ip rtp priority**, **frame-relay priority-group**, **frame-relay qos-autosense**, **ip rtp priority**, **max-reserved-bandwidth**, **show interfaces fair-queue**, **show interfaces random-detect**, **show queue**, **show queueing**, **show traffic-shape**, **show traffic-shape queue**, **show traffic-shape statistics**.
Legacy QoS Command Deprecation: Removed Commands	Cisco IOS XE Release 3.2S	The legacy QoS commands were removed. This means that you must use the appropriate replacement MQC commands. The following commands were removed: **custom-queue-list**, **fair-queue (WFQ)**, **frame-relay adaptive-shaping** (**becn** keyword), **frame-relay adaptive-shaping** (**foresight** keyword), **frame-relay bc**, **frame-relay be**, **frame-relay cir**, **frame-relay congestion threshold de**, **frame-relay congestion threshold ecn**, **frame-relay custom-queue-list**, **frame-relay fair-queue**, **frame-relay fecn-adapt**, **frame-relay ip rtp priority**, **frame-relay priority-group**, **frame-relay qos-autosense**, **ip rtp priority**, **max-reserved-bandwidth**, **show interfaces fair-queue**, **show interfaces random-detect**, **show queue**, **show queueing**, **show traffic-shape**, **show traffic-shape queue**, **show traffic-shape statistics**.



CHAPTER 8

QoS Packet Marking

QoS Packet Marking refers to changing a field within a packet either at Layer 2 (802.1Q/p CoS, MPLS EXP) or Layer 3 (IP Precedence, DSCP and/or IP ECN). It also refers to preserving any classification decision that was reached previously.

About

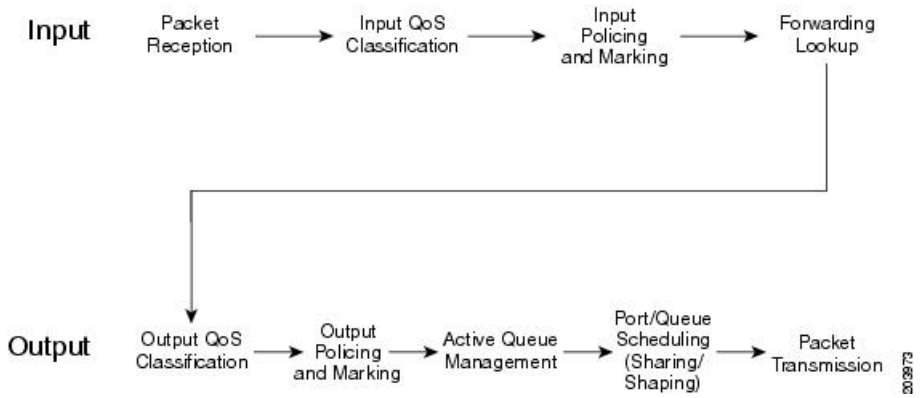
Marking Definition

Marking is similar conceptually to "service class" designation on an airplane ticket: first, business, or economy. This value reflects the level (quality) of service you should receive. Similarly, we mark a value in a packet to indicate the service class (henceforth termed service-class) for that packet as it traverses the network. By examining the marked value, network elements can decide how to treat your packet.

People in business-class may have used a variety of means to achieve that designation. They may have paid extra, used airmiles, or been lucky and booked at the normal rate when no other seat was available. Elsewhere, someone performed the complex task of classification - determining eligibility for a particular service-class then marked the ticket with a mere designation: first-class, business-class, or economy-class. Flight-attendants are unconcerned with how eligibility was determined; they simply look at the class marked on the ticket and provide that level of service.

This dynamic plays out in the networking world. One device may perform complex classification on the data in a flow, determining an appropriate service-class. Other network elements "trust" the value marked in packets they receive and provide service appropriate for that designation.

Figure 55: QoS Packet Processing



Within the context of QoS packet processing, marking occurs after classification and before queuing and is applicable on ingress or egress.

Typically, you would create a *trust boundary* at the edge of the network, then classify and mark packets on the edge device. Then, you would use that marked field for classification and determination of per-hop treatment throughout the network.

Note A trust boundary enables you to apply network-controlled marking on all packets as they enter the network and to remove or modify any non-default markings you did not apply.

Imagine that your system recognizes router ports with attached VoIP devices. You could mark the differentiated services code point (DSCP) value of voice packets as EF (at the edge of the network) and employ DSCP-based classification throughout the network to determine those packets that warrant low latency treatment.

Why Mark Packets

Reasons for marking packets include the following:

- Indicate the treatment you would like a packet to receive as it traverses the network.
- Perform complex classification once. By marking the service class, you can use simpler, less cpu-intensive classification elsewhere in the network.
- Perform classification at a point in the network where you have greater visibility into the flow. For example, if data is encrypted, you cannot perform complex classification such as determining the application carried within that flow. Instead, you could classify prior to encryption and mark a value in the unencrypted header that is visible to network elements along the path.

  As a packet traverses networks managed by different autonomous entities (e.g., the service provider network between two enterprise offices), you may need to re-mark if the markings to service-level designations are inconsistent across those networks.

  As a packet traverses different networking technologies the fields available to indicate service-class may differ. For example, you might carry service-class designation in the DSCP field of an IP packet but if this packet traverses an the multiprotocol label switching (MPLS) network only the MPLS experimental (EXP) field may be usable by network elements to determine service-class. As you enter that portion of the network, you may need to determine the appropriate marking of the MPLS EXP bits.

As a network operator you may contract to accept data from a user at a certain rate. Rather than dropping packets that exceed that rate, you can mark them as a lesser service-class.

Approaches to Marking Packets

You have two main approaches to marking packets: the **set** command and a policer marking action.

Note We only briefly touch upon "policing" actions within this chapter.

set Command

The simplest approach to marking packets on a router is to use the **set** command in a *policy-map* definition. (A policy-map is where you specify a QoS action for each class of traffic that you have defined).

You may decide to classify all RTP ports into a traffic class and mark each packet with AF41. If so, the policy-map may look something like this:

```
policy-map mark-rtp
  class rtp-traffic
    set dscp af41
```

Policer Marking Action

Recall that you can use a policer to drop packets within a traffic class above a defined rate. Alternatively, you could mark packets above that rate and allow them to receive a different per-hop treatment than packets below that rate.

For example, let's say that video traffic arrives at your router marked AF41. You may decide to consider user traffic up to 2 Mbps top *assured forwarding behavior* and to demote any traffic exceeding 2Mbps to AF42 (and considered *out of contract* - non-conforming).

The policy-map might appear as follows:

```
class-map video-traffic
  match dscp af41
!
policy-map enforce-contract
  class video-traffic
    police cir 2m conform-action transmit exceed-action set-dscp-transmit AF42
```

Scope of Marking Action

Similar to classification, marking cannot access every field within a data packet. For example, if an IP packet is encapsulated in multiprotocol label switching (MPLS), it cannot mark the DSCP within the IP header as that would require first de-capsulating from MPLS. However, you could mark the MPLS experimental (EXP) bits.

Note Only Layer 2 and outer Layer 3 headers are available for marking.

Multiple Set Statements

You can configure multiple marking rules within a single class (or policer action). This allows you to mark both Layer 2 and Layer 3 fields within the same packet, or if multiple traffic types are present in the same class, define marking values for each type.

For example consider the following egress policy attached to an Ethernet subinterface:

```
policy-map mark-rtp
  class rtp-traffic
    set cos 4
    set mpls exp topmost 4
    set dscp af41
```

If an MPLS packet were forwarded through this subinterface, the Layer 2 COS field and the EXP bits in the MPLS header would be marked. If an IP datagram were encapsulated in that packet, its DSCP value would remain unchanged. However, if an IP packet were forwarded through the subinterface, its Layer 2 COS value and Layer 3 DSCP values would be marked.

For details, refer to the command pages for set cos, on page 160, set mpls experimental topmost, on page 164, and set dscp, on page 161.

Marking Internal Designators

Cisco routers allow you to mark two internal values (qos-group and discard-class) that travel with the packet within the router but do not modify the packet's contents.

Typically, you mark these in an ingress policy and use them to classify to a traffic-class or WRED drop profile in an egress policy. For example, you may want to base your egress classification on a user's IP address but realize that encryption is configured and the user's IP address is invisible on an egress interface. You could classify their traffic on ingress (before encryption) and set an appropriate qos-group value. On egress, you could now classify based on the qos-group and choose the action accordingly.

Ingress vs. Egress Marking Actions

Certain marking values are only relevant to ingress or egress policies. For example, marking the ATM CLP bit or Frame Relay DE bit in an ingress policy is meaningless as they are discarded when the packet is decapsulated. Similarly, marking qos-group or discard-class in an egress policy is unproductive as these leave the packet unchanged and are discarded when we enqueue the packet for forwarding to the next hop.

Imposition Marking

Under special circumstances, you can mark a header field that has not yet been added to a packet (we term this behavior *imposition marking*).

The most common example of imposition marking is the application ofthe **set mpls experimental imposition** command - you can use it on an ingress interface where a packet may arrive containing an IP datagram and no multiprotocol label switching (MPLS) header. When and if the router encapsulates the datagram with a MPLS header, the EXP bits will be marked accordingly as specified by this command.

Application of the **set dscp tunnel** and **set precedence tunnel** commands (for IPv4 only) represent another example of imposition marking. If an egress policy is applied on a tunnel interface, no tunnel header exists when the policy executes. This means that any marking would apply to the original (eventually inner) IP

header. Using either command, you can mark the tunnel (outer) IP header and leave the original header unchanged.

The following table lists the tunnel types and encapsulation variants that support these commands:

When a new header is added (encapsulated), any QoS marking in the inner header is copied to the outer header. For example, when an IP datagram is encapsulated with an MPLS header, the default behavior is to copy the IP Precedence bits from the IP header to the MPLS EXP bits in the newly-imposed header.

Regarding header disposition, we typically do not copy any outer marking(s) to the inner header. For example, at the endpoint for a GRE tunnel, let's say that we receive a packet with different DSCP values in the outer and inner IP headers. When we remove the outer header we do not copy its DSCP value to the inner header.

Configuration Examples

Example 1: Configuring Ingress Marking

You can set up a trust boundary at the edge of a network (where marking is used) to indicate service-class for some traffic and to bleach all other traffic (see *** below). Enforcing a trust boundary at all ingress ports to the network allows you to maintain control of which applications are mapped to each service-class within the network:

```
policy-map ingress-marking
  class voice
    set dscp ef
  class video
    set dscp af41
  class scavenger
    set dscp cs1
  class class-default                                    ***
    set dscp 0                                           ***
!
interface gigabitethernet1/0/0
  Service-policy in ingress-marking
```

For details, refer to the page set dscp, on page 161.

Example 2: Configuring Egress Marking

If a different administrator controls a portion of a network path and uses a different DSCP to service-class mapping, egress marking may be necessary (e.g., within your enterprise, you classify 12 distinct classes of traffic as described in RFC4594). However, your service provider only provides a three-class model.

You may also need egress marking to indicate treatment for certain classes in a Layer 2 network (like Ethernet, frame-relay, or ATM switched networks):

```
policy-map egress-marking
  class scavenger
    set atm-clp
```

For command details, refer to the page set atm-clp, on page 160.

Example 3: Configuring MPLS EXP Imposition

With MPLS, a provider edge (PE) router encapsulates datagrams or frames with MPLS headers. Switching decisions within the core are based on the MPLS headers without visibility into the encapsulated data.

Consider a Layer 3 MPLS network where IPv4 datagrams are encapsulated in MPLS headers. On the customer edge (CE) facing interface we have visibility into the IPv4 header of the packet. On the core-facing interface, we have encapsulated datagrams with MPLS headers and we cannot see beyond those headers.

By default, we copy the IP precedence to the MPLS EXP bits. What if we want to override this behavior? We can't parse the IPv4 type of service byte on the core-facing interface. We can, however, parse the IP header on ingress and store the EXP value we plan to set when MPLS headers are added. Although MPLS headers are absent when we execute the command, the router retrieves the instruction and marks the EXP bits on the egress interface:

```
policy-map mpls-exp-remark
  class voice
    set mpls experimental imposition 5
  class video
    set mpls experimental imposition 4
  class scavenger
    set mpls experimental imposition 0
!
interface gigabitethernet1/0/0
  policy-map input mpls-exp-remark
```

For command details, refer to the page set mpls experimental imposition, on page 164.

Example 4: Configuring Tunnel Imposition Marking

Conceptually, tunnel and MPLS EXP imposition marking are similar. We want to mark a value in a header that has not yet been added to the packet and with a Layer 3 tunneling technology like GRE or IPinIP, a Layer 3 datagram may be encapsulated with an outer IP header. (Refer to Imposition Marking, on page 144.)

Let's say that we have a DMVPN network where a branch location encrypts data and encapsulates it with a GRE header before sending it over a public IP network. An administrator may attach a policy-map to the tunnel interface to prioritize applications within that tunnel and may also need to mark the DSCP of the outer IP header to indicate service-class within the provider's network. When the policy is executed, the outer header has not yet been added and commands like **set dscp** or **set precedence** would mark the inner IP header.

To solve the problem, we use the **set dscp tunnel** and **set precedence tunnel** commands, as they allow you to set the value in an outer header that has not yet been added.

In the following example, voice and video traffic are classified and queued separately within the enterprise network. The service provider has a smaller number of service-classes and we have decided to put both voice and video into the priority class within the provider's network.

By marking the DSCP in the outer tunnel header we achieve this yet preserve original markings in the inner header:

```
policy-map mark-outer-gre-header
  class voice
    priority level1 percent 20
    set dscp tunnel ef
  class video
    priority level 2 percent 20
    set dscp tunnel ef
```

```
!
interface tunnel100
  service-policy out mark-outer-gre-header
```

For command details, refer to the page set dscp tunnel, on page 162.

Example 5: Configuring QoS-Group Marking

Occasionally, you may want to base egress queuing on ingress classification. For example, let's say you want more than 8 egress queues on a MPLS-enabled interface. Using egress classification, you are limited to MPLS EXP bits and therefore 8 classes. As a solution, you could perform classification on the ingress interface and set a QoS group for packets that match that classification. QoS group has relevance only within the current router; it doesn't alter anything in the packet header. Instead, it's a value associated with the packet as it passes through the router.

In the following example we use Network Based Application Recognition (NBAR) classification on ingress and mark both telepresence and jabber video with qos-group 4. In the egress policy we classify based on the qos-group we marked on ingress (see "***"):

```
class-map telepresence-video
  match protocol telepresence-media
class-map jabber-video
  match protocol cisco-jabber-video
class-map egress-video-traffic                          ***
  match qos-group 4                                     ***
!
policy-map mark-qos-group
  class telepresence-video
    set qos-group 4
  class jabber-video
    set qos-group 4
!
policy-map egress-queuing
  class egress-video-traffic
    bandwidth remaining percent 50
!
interface gig 1/0/0
  service-policy in mark-qos-group
!
interface serial1/1/0
  service-policy out egress-queuing
```

For command details, refer to the page set qos-group, on page 165.

Example 6: Configuring Discard-Class Marking

In Example 5: Configuring QoS-Group Marking, on page 147, we marked both telepresence video and jabber video with qos-group 4 and placed both of these applications into the same egress queue.

What if we want to run Weighted Random Early Detection (WRED) on the egress queue and drop the jabber video first during congestion. Typically, WRED examines the precedence or DSCP value to determine drop thresholds for a flow. However, as indicated in Example 3: Configuring MPLS EXP Imposition, on page 146, we do not have visibility into the IP header. A solution is to mark a second internal value named discard-class. Then, we could use the qos-group to select the egress class (and queue) and the discard-class to select the WRED drop profile within that class.

```
class-map telepresence-video
  match protocol telepresence-media
class-map jabber-video
  match protocol cisco-jabber-video
class-map egress-video-traffic
  match qos-group 4
!
policy-map mark-qos-group
  class telepresence-video
    set qos-group 4
    set discard-class 1
  class jabber-video
    set qos-group 4
    set discard-class 2
!
policy-map egress-queuing
  class egress-video-traffic
    bandwidth remaining percent 50
    random-detect discard-class-based
    random-detect discard-class 1 24 40
    random-detect discard-class 2 22 30
!
interface gig 1/0/0
  service-policy in mark-qos-group
!
interface serial1/1/0
  service-policy out egress-queuing
```

For command details, refer to the page set discard-class, on page 161.

Verifying QoS Packet Marking

The **show policy-map interface** command is the primary means of verifying any QoS behavior on IOS XE platforms. Although the packet forwarding path (dataplane) is separated from the IOS instance (control plane), statistics are still reported through this well-known IOS command. This functionality is enabled by default.

This table describes the fields we employ in the following sections.

Table 5: show policy-map interface Field Descriptions (those useful for verifying marking)

Field	Description
Service-policy input	Denotes the name of the input service policy applied to the specified interface or VC
Class-map	Specifies the class of traffic being displayed. Output is displayed for each configured class in the policy. The choice for implementing class matches (e.g., match-all or match-any) can also appear adjacent to the traffic class
packets, bytes	Specifies the number of packets (shown in bytes) identified as belonging to the class of traffic being displayed
offered rate	Specifies the rate in bits per second of the packets entering the class
Match	Specifies the match criteria for the traffic class
QoS Set	Details the QoS marking actions configured for the particular class

Field	Description
Packets marked	If enabled, denotes the total number of packets marked for the particular class. If not enabled, you see "Marker statistics: Disabled."

Verifying with the show policy-map interface Command

The **show policy-map interface** command is the primary means of verifying any QoS behavior on IOS XE platforms. Ordinarily, knowing how many packets match a particular class ("class match statistics, " which is enabled by default) and what (if any) marking action is configured suffices to know how many packets were marked by that action.

Note You should understand how *class match statistics* (enabled by default) and *marking statistics* (disabled by default) differ. Typically, the former is sufficient. When a packet "hits" a class, you can assume it is marked. However, if you configure multiple, mutually exclusive marking values, and need to know how many packets were marked with each **set** command, you can enable marking statistics with all its caveats.

Here is an example of ingress marking with a policy attached to a physical interface. In this example, let's say that jabber-video is configured on ports 2000-3000:

```
class-map match-all jabber-video
  match ip rtp 2000 3000
!
policy-map mark-traffic
  class jabber-video
    set dscp af41

show policy-map int g1/0/0
GigabitEthernet1/0/0

  Service-policy input: mark-traffic

    Class-map: jabber-video (match-all)
      850 packets, 51000 bytes                                    note 1
      5 minute offered rate 2000 bps, drop rate 0000 bps
      Match: ip rtp 2000 3000
      QoS Set                                                     note 2
        dscp af41
          Marker statistics: Disabled

    Class-map: class-default (match-any)                          note 3
      0 packets, 0 bytes
      5 minute offered rate 0000 bps, drop rate 0000 bps
      Match: any
```

Footnotes

note 1	Statistics for the class match
note 2	Packet matching section

note 3	Class-Default statistics section

Observe "`Marker statistics: Disabled`" in the output of ingress marking. If you are invoking multiple statistics and find the information provided in the previous output insufficient, you can enable "Packet Marker Statistics."

Verifying with QoS Packet Marking Statistics

Before you begin

Either

- Remove all policy-maps, issue the command, and re-attach all policy-maps.
- Issue the command, save the configuration, and reload the router.

Note Enabling QoS: Packet Marking Statistics may increase CPU utilization on a scaled configuration. Weigh the benefits of displaying statistics information against the increased CPU utilization for your system.

Enabling QoS Packet Marking Statistics

To enable Packet Marking Statistics, issue the **platform qos marker-statistics** command in configuration mode.

Displaying QoS Packet Marking Statistics

To display the packet statistics of all classes that are configured for all service policies either on the specified interface (or subinterface) or on a specific Permanent Virtual Circuit (PVC), use the **show policy-map interface** command.

When we singularly-configure marking in a policy-map, the output from an ASR 1000 Series Aggregation Services Router would appear as follows:

```
policy-map remark-af41
  class af41-traffic
    set dscp tunnel ef
```

Let's place this map on a tunnel interface with traffic marked af41 in the user's IP header and DSCP marked EF in the GRE IP header. The output of the **show policy-map interface** will appear as follows:

```
show policy-map interface tunnel1

  Service-policy output: remark-af41

    Class-map: af41-traffic (match-all)
      978 packets, 68460 bytes                                  note 1
      5 minute offered rate 2000 bps, drop rate 0000 bps
      Match:  dscp af41 (34)
      QoS Set                                                   note 2
        dscp tunnel ef
          Marker statistics: Disabled                           note 3
```

```
  Class-map: class-default (match-any)
    365 packets, 25550 bytes
    5 minute offered rate 0000 bps, drop rate 0000 bps
    Match: any
```

Footnotes

note 1	Displays the class match statistics (assume all "observed" packets are marked AF41).
note 2	Marking is the only action configured.
note 3	Per-set action statistics are disabled by default.

Now, if we enable marking statistics, output from the **show policy-map interface** command would appears as follows:

```
show policy-map interface tunnel1

  Service-policy output: remark-af41

    Class-map: af41-traffic (match-all)
      575 packets, 40250 bytes
      5 minute offered rate 1000 bps, drop rate 0000 bps
      Match:  dscp af41 (34)
      QoS Set
        dscp tunnel ef
          Packets marked 575                  note

    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0000 bps, drop rate 0000 bps
      Match: any
```

Footnote

note	We have now enabled marking statistics but in this example the information is redundant.

For command details, refer to the page set dscp tunnel, on page 162.

Validating the Dataplane Configuration

To verify that the dataplane configuration reflects the IOS control plane configuration, use the **show platform hardware qfp active feature qos interface [input|output]** command, which engages only if issued before you attach any policy-map to an interface. So, you must do one of the following:

- Remove all policy-maps, issue the command and re-attach all policy-maps.
- Issue the command, save the configuration and reload the router.

In the following output, notice that we have configured the actions and set the values on the dataplane:

```
show platform hardware qfp active feature qos interface g1/0/0 input
```

```
Interface: GigabitEthernet1/0/0, QFP interface: 12
  Direction: Input
    Hierarchy level: 0
    Policy name: mark-traffic
      Class name: jabber-video, Policy name: mark-traffic
        QOS Set:
            dscp 34                                                    note
      Class name: class-default, Policy name: mark-traffic
```

Footnote

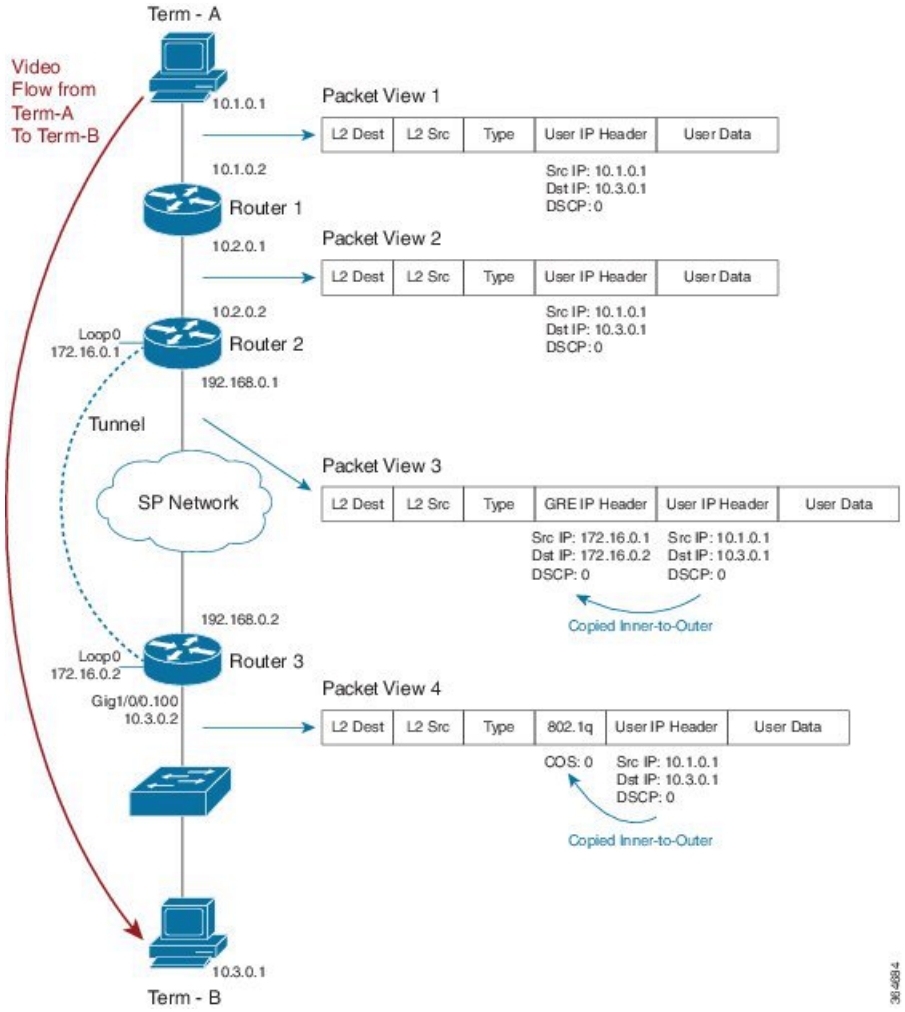
note	Dataplane is programmed to mark.

Network-Level Configuration Examples

In the scenarios that follow, a video-flow transits from Terminal-A to Terminal-B.

Example 1: Propagating Service-Class Information Throughout the Network

Figure 56: Propagating Service-Class Information Throughout the Network

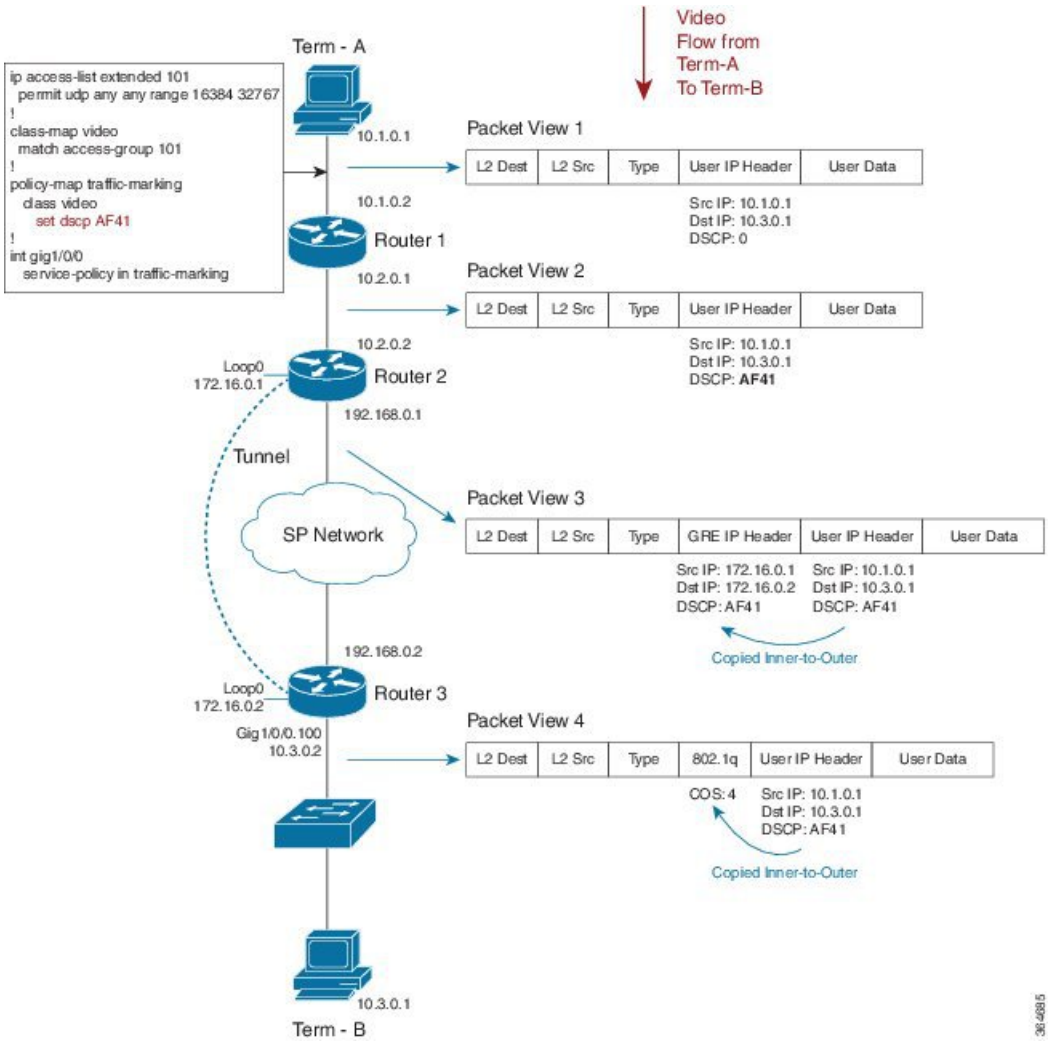


Imagine that an application marks the video stream with DSCP codepoint 0 (see Packet View 1). To cross the provider's network, we send the stream through a GRE tunnel (possibly encrypted). Packet View 3 shows that we have encapsulated the users' IP datagram in a GRE packet. Notice how the DSCP codepoint is copied by default to the imposed GRE header.

With the last hop at the final destination, Router 3 sends a VLAN tagged packet to a switch (see Packet View 4). Observe that the GRE header was stripped and a Dot1Q header was added due to the VLAN configuration. The precedence portion of the user's DSCP 0 (000 000) is copied by default to the COS bits of the VLAN header. The COS value set is 0 (000).

Example 2: Indicating Service-Class by Marking at the Network's Edge

Figure 57: Indicating Service-Class by Marking at the Network's Edge



In this example, we modify the default behavior by remarking the DSCP of users' traffic in an ingress policy as it enters Router 1. The following code shows how we do this:

```
ip access-list extended 101
 permit udp any any range 16384 32767
!
class-map video
 match access-group 101
!
policy-map traffic-marking
  class video
    set dscp AF41
!
int gig1/0/0
  service-policy in traffic-marking
```

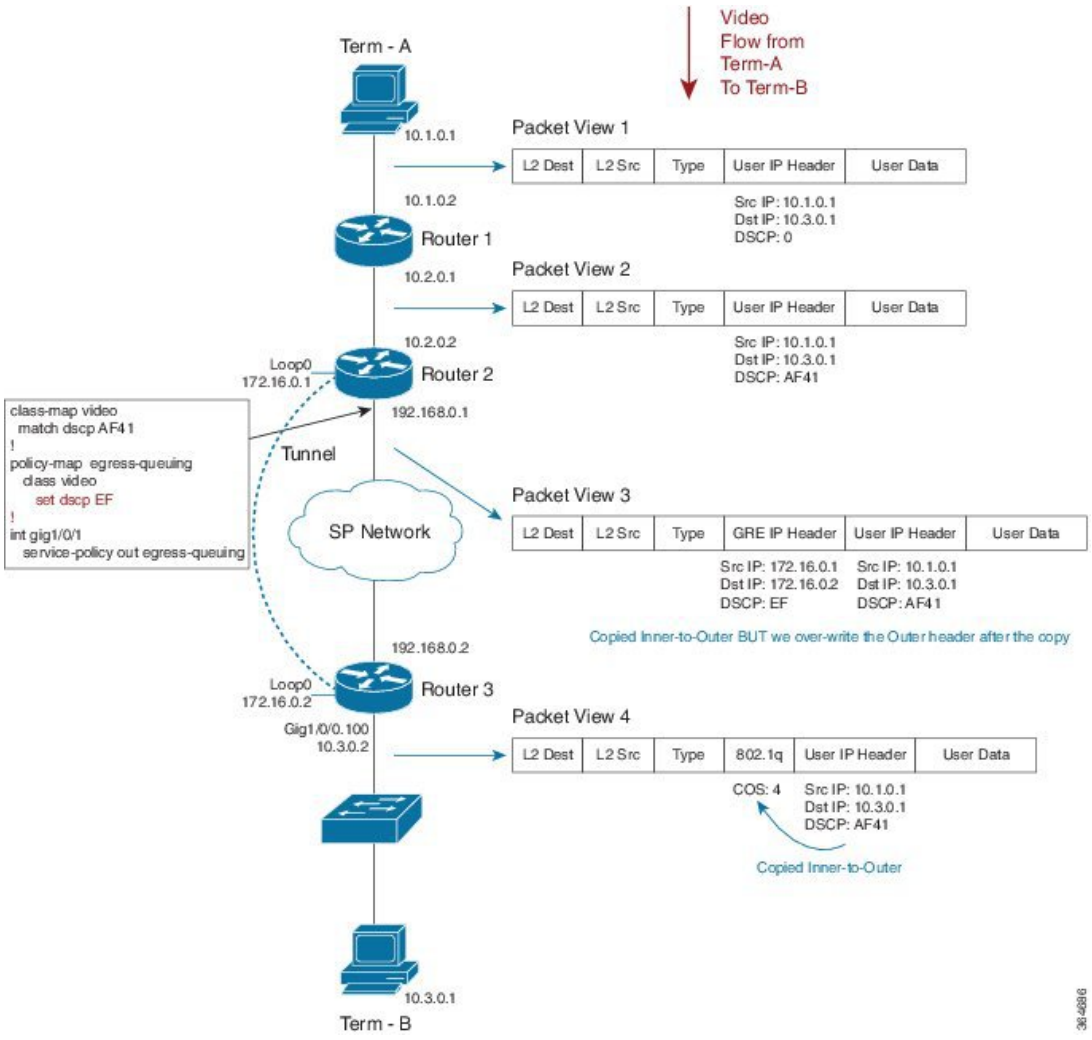
Let's say that we designate video traffic as DSCP AF41 throughout the network. When the packet reaches the GRE interface on egress, its DSCP value has already been changed to AF41 and its behavior matches that in Example 1. We send the stream through a GRE tunnel (possibly encrypted) as it traverses the providers network. Notice how the newly-marked DSCP codepoint (AF41) is copied by default to the imposed GRE header.

When we arrive at our destination, the router sends a VLAN-tagged packet to the last hop (a switch). The precedence portion of the users' DSCP value is copied by default into the COS bits of the VLAN header. As our DSCP is now AF41 (100 010), the COS value will be 4 (100).

For command details, refer to the command page set dscp, on page 161.

Example 3: Remarking Traffic to Match Service Provider Requirements

Figure 58: Remarking Traffic to Match Service Provider Requirements



In this example, we mark the DCSP value within the network while the service provider anticipates a different marking. The following code shows how we handle this:

```
class-map video
  match dscp AF41
!
policy-map  egress-queuing
  class video
    set dscp EF
!
int gig1/0/1
  service-policy out egress-queuing
```

We mark DCSP as AF41 for video within out network while the service provider expects video packets to be marked EF. On the egress Gig interface of Router 2, we add a policy that contains queuing commands (recall that we are only focusing on the marking portion of the configuration in this example).

When the packet reaches the egress physical interface it already has the GRE header imposed and we copy the DSCP value of AF41 from the inner encapsulated datagram. The policy on the physical interface changes the DSCP value in the outer GRE header only.

Note Notice how the inner-user datagram IP header is unchanged.

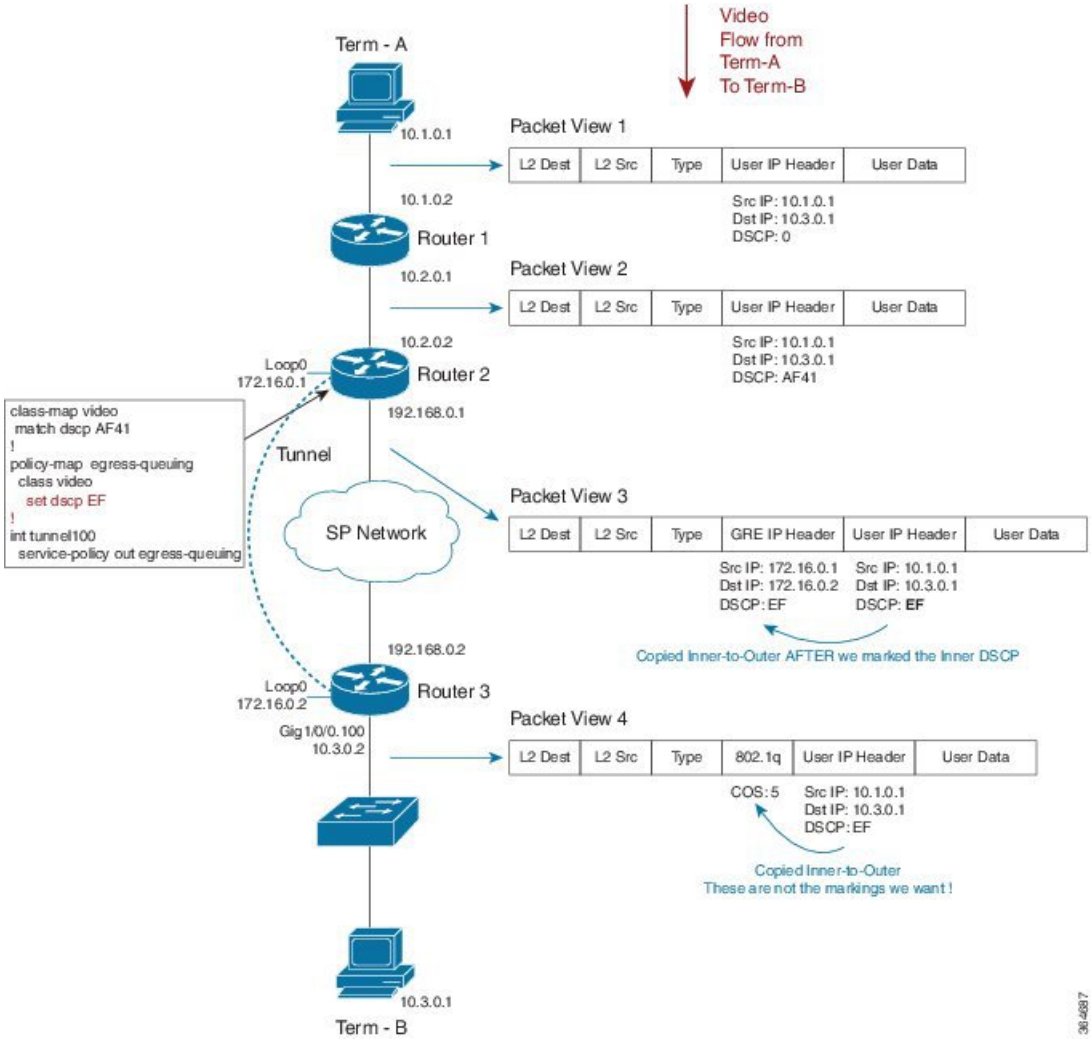
When we reach Router 3 and exit the tunnel, the tunnel GRE header is stripped. Henceforth, only the user datagram IP header is visible, still preserving the AF41 value we marked on ingress to the network.

As in previous examples, the router sends a VLAN-tagged packet to the last hop (a switch). By default, the precedence portion of the User IP Header's DSCP value is copied into the COS bits of the VLAN header (802.1q). As the DSCP value is currently af41 (100 010), the COS value will be 4 (100).

For command details, refer to the page set dscp, on page 161.

Example 4: Remarking on a Tunnel Interface for an SP Network - Potential Gotcha

Figure 59: Remarking on a Tunnel Interface for an SP Network - Potential Gotcha



In this example, we place the QoS policy on the tunnel interface of Router 1 rather than on the physical interface. (There are many advantages to configuring queuing per tunnel rather than as an aggregate policy on the physical interface.) The following code shows how we do this:

```
class-map video
  match dscp AF41
!
policy-map  egress-queuing
  class video
    set dscp EF
!
int tunnel100
```

```
service-policy out egress-queuing
```

We focus solely on the marking portion of the policy. The key point is that marking on the tunnel interface is performed before the tunnel headers are added.

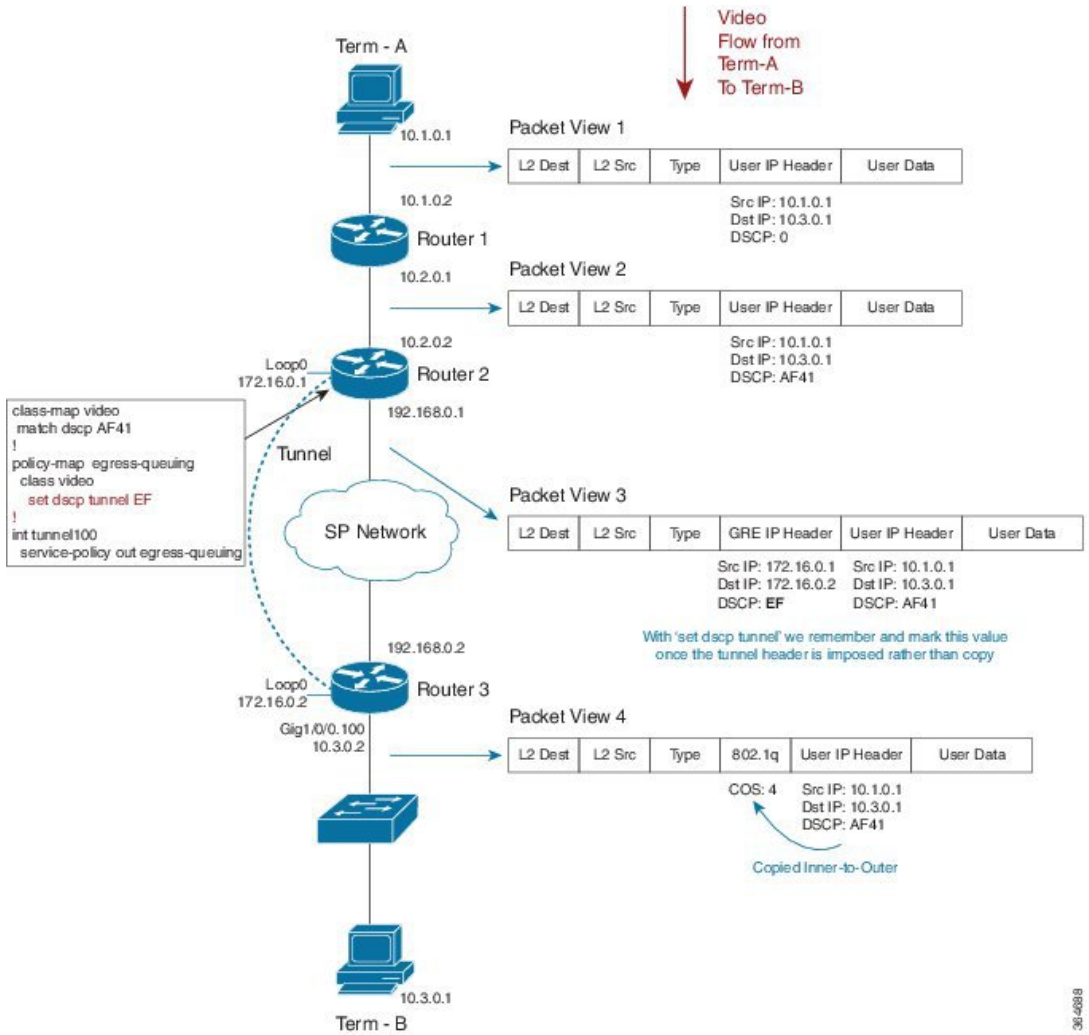
Notice how our policy has over-written the DSCP in the user datagram IP header. Because this happened before GRE encapsulation, we copy the newly-marked value to the outer header.

When we reach Router 3 and exit the tunnel the tunnel GRE header is stripped. Because we marked the user datagram header, the new value propagates through the rest of the network. This is not the behavior we wanted.

For command details, refer to the page for set dscp, on page 161.

Example 5: Using Tunnel Imposition Marking to Remark for an SP Network

Figure 60: Using Tunnel Imposition Marking to Remark for an SP Network



In this example, we use the **set dscp tunnel** *dscp-value* command to alter only the tunnel IP Header:

```
class-map video
  match dscp AF41
!
policy-map  egress-queuing
  class video
    set dscp tunnel EF
!
int tunnel100
  service-policy out egress-queuing
```

We have a QoS policy on the tunnel interface of Router 2 and we have used the **set dscp tunnel** command rather than **set dscp** command.

We have yet to impose the GRE header. The **set dscp tunnel** command dictates that we remember the DSCP value; during encapsulation we use this value instead of copying "inner to outer." Observe that the DSCP value in the users IP datagram header is unchanged. The **set dscp tunnel** command will alter only the tunnel IP header.

For command details, refer to the page for set dscp tunnel, on page 162.

Command Reference

platform qos marker-statistics

To enable individual statistics collection for each marking action in every policy configured on the router, use the **platform qos marker-statistics** command in global configuration mode. To disable packet marking statistics, use the **no** form of this command.

[no] platform qos marker-statistics

Syntax Description

This command has no arguments or keywords.

Command Default

Disabled (no packet marking statistics are displayed). The network operator relies on class match statistics.

Command Modes

policy-map (config-pmap)

Usage Guidelines

This command executes only if issued before any policy-map is attached to an interface. So, you must do one of the following:

- Remove all policy-maps, issue the command and re-attach all policy-maps.
- Issue the command, save the configuration and reload the router.

Note Enabling packet marking statistics may increase CPU utilization on a scaled configuration. So, weigh the benefits of the statistics information against the increased CPU utilization for your system.

set atm-clp

To set the ATM cell loss priority (CLP) bit, use the **set atm-clp** command in policy-map class configuration mode. To disable this setting, use the **no** form of this command.

[no] set atm-clp

Syntax Description

This command has no arguments or keywords.

Command Default

The ATM CLP bit is not set.

Command Modes

policy-map (config-pmap)

Usage Guidelines

On ATM interfaces, you can use the **set atm-clp** command in an outbound policy to set the ATM-CLP bit in ATM cell headers to 1.

This command is supported for ATM, PPPoA, PPPoEoA and L2TPv3 encapsulations. It is not supported if the policy is attached to a tunnel rather than directly to the VC.

You cannot attach a policy-map containing ATM set cell loss priority (CLP) bit QoS to PPP over X (PPPoX) sessions. The map is accepted only if you do not specify the **set atm-clp** command.

For an example using the **set atm-clp** command to configure egress marking, please refer to Example 2: Configuring Egress Marking, on page 145.

set cos

To set the Layer 2 class of service (CoS) value of an outgoing packet, use the **set cos** command in policy-map class configuration mode. To disable this setting, use the **no** form of this command.

[no] set cos *cos-value*

Syntax Description

cos-value	Specifies the IEEE 802.1Q CoS value of an outgoing packet ranging from 0 to 7

Command Default

Either IP Precedence or MPLS EXP bits are copied from the encapsulated datagram.

Command Modes

policy-map (config-pmap)

Usage Guidelines

You can use the **set cos** command to propagate service-class information to a Layer 2 switched network. Although a Layer 2 switch may not be able to parse embedded Layer 3 information (such as DSCP), it might be able to provide differentiated service based on CoS value. Switches can leverage Layer 2 header information, including the marking of a CoS value.

Traditionally the **set cos** command had meaning only in service policies that are attached in the egress direction of an interface because routers discard Layer 2 information from received frames. With the introduction of features like EoMPLS and EVC, the setting of CoS on ingress has meaning, such that you can preserve Layer 2 information throughout the routed network.

set cos-inner

To set the Layer 2 CoS value in the inner VLAN tag of a QinQ packet, use the **set cos-inner** command in policy-map class configuration mode. To disable this setting, use the **no** form of this command.

[no] set cos-inner *cos-value*

Syntax Description

cos-value	Specifies a IEEE 802.1q CoS value ranging from 0-7

Command Default

Either IP Precedence or MPLS EXP bits are copied from the encapsulated datagram.

Command Modes

policy-map (config-pmap)

Usage Guidelines

Traditionally, because routers discard Layer 2 information from received frames, the **set cos-inner** command had meaning only in service policies that are attached in the egress direction of an interface. With the introduction of features like EoMPLS and EVC, the setting of CoS on ingress has significance as you can preserve Layer 2 information throughout the routed network.

set discard-class

To set the QoS discard class for a packet, use the **set discard-class** command in policy-map configuration mode. To disable this setting, use the **no** form of this command.

[no] set discard-class *discard-class-value*

Syntax Description

discard-class-value	Specifies a Discard Class value ranging from 0 to 7

Command Default

The discard-class value associated with a packet is set to 0.

Command Modes

policy-map (config-pmap)

Usage Guidelines

The **set discard-class** command allows you to associate a discard class value with a packet while processed by the router. Setting this value leaves the packet unchanged.

You can use the discard class and discard-class based WRED in egress policies to control which packets are dropped during congestion.

set dscp

To set the DSCP value in the IP header, use the **set dscp** command in policy-map class configuration mode. To disable this setting, use the **no** form of this command.

[no] set dscp *dscp-value*

Syntax Description

dscp-value	Sets the DSCP value in an IP header ranging from 0 to 63. You can specify the value numerically or by using it's well known DiffServe name (e.g., EF)

Command Default

Retain the existing DSCP value in the received packet.

Command Modes

policy-map (config-pmap)

Usage Guidelines

The command may be used in ingress or egress policies.

You can use the DSCP value to indicate the QoS treatment a packet should receive as it traverses a network.

Note The differentiated services architecture using DSCP supersedes use of precedence.

This command marks packets where the outermost Layer 3 header is either IPv4 or IPv6.

If issued in an egress policy-map, this command will not alter the class or queue selection but might influence the WRED drop profile selection.

The **set dscp** and **set ip dscp** commands behave identically, marking both IPv4 and IPv6 packets.

Note This differs from the process of classification wherein the **match ip dscp** command classifies only IPv4 packets while the **match dscp** command classifies both IPv4 and IPv6 packets.

set dscp tunnel

To set the DSCP value in a tunnel header that has not yet been added to a packet, use the **set dscp tunnel** command in policy-map class configuration mode. To disable this setting, use the **no** form of this command.

[no] set dscp tunnel *dscp-value*

Syntax Description

dscp-value	Specifies the DSCP value in a tunnel header ranging from 0 to 63. You can either specify the value numerically or use its well known DiffServe name (e.g. EF).

Command Default

DSCP value from an encapsulated datagram is copied to the newly-imposed tunnel header.

Command Modes

policy-map (config-pmap)

Usage Guidelines

This command only makes sense before a tunnel header is added.

Note You can use this command in either an ingress or egress policy that is attached to a tunnel interface. However, if the latter is attached, the command has no meaning because all headers would be added when the policy is evaluated.

On the Cisco ASR Series Aggregation Services Router, the **set dscp tunnel** command is supported for IPv4 only. See Imposition Marking, on page 144 for a table that lists the supported DSCP tunnel marking configurations.

For an example using this command to encapsulate a Layer 3 datagram with an outer IP header, please refer to Example 4: Configuring Tunnel Imposition Marking, on page 146.

set fr-de

To set the frame-relay (FR) discard eligible (DE) bit, use the **set fr-de** command in policy-map class configuration mode. To disable the setting, use the **no** form of this command.

[no] set fr-de

Syntax Description This command has no arguments or keywords.

Command Default The DE bit is not set when datagrams are encapsulated with frame relay.

Usage Guidelines On serial interfaces configured with Frame Relay encapsulation, you can use the **set fr-de** command in an outbound policy to set the Discard Eligible bit in the Frame Relay header to 1.

set ip dscp

To preserve backwards compatibility, we support two command variants that perform identical functions: **set ip dscp** and **set dscp**. You can use either to mark the DSCP value in the IP header. Please refer to the **set dscp** command page (set dscp, on page 161) for more information.

set ip dscp tunnel

To preserve backwards compatibility, we support two command variants that perform identical functions: **set ip dscp tunnel** and **set dscp tunnel**. Please refer to the **set dscp tunnel** command page (set dscp tunnel, on page 162) for details.

set ip precedence

To preserve backwards compatibility, we support two command variants that perform identical functions: **set ip precedence** and **set precedence**. You can use either to mark the precedence value in the IP header. Please refer to the **set precedence** command page (set precedence, on page 164) for more information.

set ip precedence tunnel

To preserve backwards compatibility, we support two command variants that perform identical functions: **set ip precedence tunnel** and **set precedence tunnel**. Please refer to the **set precedence tunnel** command page (set precedence tunnel, on page 165) for more information.

set mpls experimental imposition

To set the value of the MPLS EXP field on all imposed label entries, use the **set mpls experimental imposition** command in policy-map class configuration mode. To disable this setting, use the **no** form of this command.

[no] set mpls experimental imposition *mpls-exp-value*

Syntax Description

mpls-exp-value	Specifies the MPLS EXP value, which ranges from 0 to 7

Command Default

MPLS value is copied from the appropriate field (usually precedence) in the encapsulated packet.

Command Modes

policy-map (config-pmap)

Usage Guidelines

The **set mpls experimental imposition** command is supported only on input interfaces. Use this command during label imposition to set the MPLS EXP field on all imposed label entries.

For an example of using this command to set the EXP bits in an MPLS header that we use to encapsulate the datagram or frame, please refer to Example 3: Configuring MPLS EXP Imposition, on page 146.

set mpls experimental topmost

To set the MPLS EXP field value in the topmost label, use the **set mpls experimental topmost** command in policy-map class configuration mode. To disable this setting, use the **no** form of this command.

[no]set mpls experimental topmost *mpls-exp-value*

Syntax Description

mpls-exp-value	Specifies the MPLS EXP value ranging from 0 to 7

Command Default

The MPLS EXP value is either copied from the innermost header on encapsulation or remains unchanged.

Command Modes

policy-map (config-pmap)

Usage Guidelines

This command marks packets provided the outermost Layer 3 header is an MPLS label when the command is evaluated.

This command sets the MPLS EXP value in the topmost label only. If multiple labels exist in a stack, the MPLS EXP value in labels other than the topmost remain unchanged.

set precedence

To set the IP Precedence value in the packet header, use the **set precedence** command in policy-map class configuration mode. To disable this setting, use the **no** form of this command.

[no] set precedence *precedence-value*

Syntax Description

precedence-value	Sets the precedence bit in the packet header, which ranges from 0 to 7

Command Default

Retain the precedence value in the received packet.

Command Modes

policy-map (config-pmap)

Usage Guidelines

The command may be used in ingress or egress policies. However, if you issue the command in an egress policy-map, it will not alter the class or queue selection but it may influence the WRED drop profile selection.

By setting a precedence value, you indicate the QoS treatment a packet should receive as it traverses a network.

Note The differentiated services architecture using DSCP largely supersedes the use of precedence.

The **set precedence** and **set ip precedence** commands behave identically, marking packets where the outermost Layer 3 header is IPv4 or IPv6. In contrast, the **match ip precedence** command classifies only IPv4 packets while the **match precedence** command classifies both IPv4 and IPv6.

set precedence tunnel

To set the IP precedence value in a tunnel header that has not yet been added to a packet, use the **set precedence tunnel** command in policy-map class configuration mode. To disable this setting, use the **no** form of this command.

[no] set precedence tunnel *precedence-value*

Syntax Description

precedence-value	Sets the precedence bit in the tunnel header ranging from 0 to 7

Command Default

DSCP (and the precedence portion) are copied from the encapsulated to the newly-imposed header.

Command Modes

policy-map (config-pmap)

Usage Guidelines

On the Cisco ASR Series Aggregation Services Router, the **set precedence tunnel** command is supported for IPv4 only. See Imposition Marking, on page 144 for a table that lists the supported DSCP tunnel marking configurations.

set qos-group

To set the QoS group identifier (ID) for a packet, use the **set qos-group** command in policy-map class configuration mode. To disable this setting, use the **no** form of this command.

[no] set qos-group *group-id*

Syntax Description

group-id	Specifies a QoS group ID ranging from 0 to 99

Command Default

QoS group-id defaults to 0.

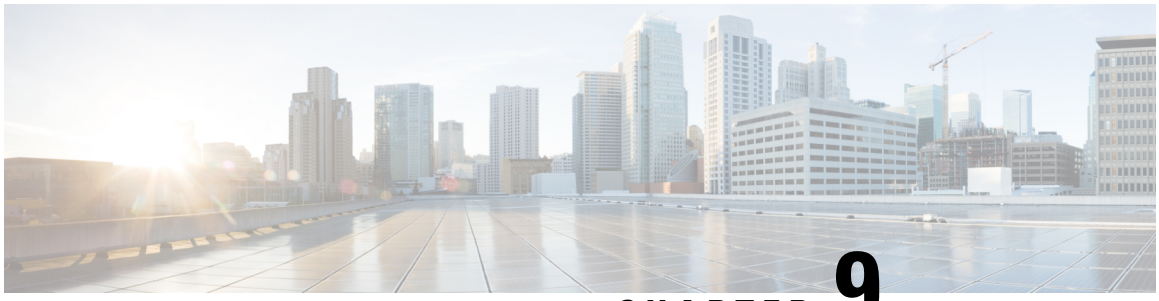
Command Modes

policy-map (config-pmap)

Usage Guidelines

The **set qos-group** command allows you to associate a group ID with a packet as it is processed by the router.

You can use the group ID in egress policies to classify packets to service-classes. Historically, this action had no meaning because we chose the service-class before egress marking occured. With color-aware policing, however, setting the QoS group ID in an egress policy can have meaning.



CHAPTER 9

QoS Packet-Matching Statistics Configuration

The QoS Packet-Matching Statistics feature comprises the following subfeatures:

- The QoS Packet-Matching Statistics: Per Filter feature allows users to count and display the number of packets and bytes matching individual filters (match statements) within a QoS class-map.
- The QoS Packet-Matching Statistics: Per ACE feature allows users to count and display the number of packets and bytes matching the individual access control entries (ACEs) in the filter.

Finding Feature Information

Your software release may not support all the features documented in this module. For the latest caveats and feature information, see Bug Search Tool and the release notes for your platform and software release. To find information about the features documented in this module, and to see a list of the releases in which each feature is supported, see the feature information table.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to https://cfnng.cisco.com/. An account on Cisco.com is not required.

Prerequisites for QoS Packet-Matching Statistics Feature

You cannot enable or disable the QoS Packet-Matching Statistics: Per Filter feature if a policy-map is associated with any interface on the system.

The QoS Packet-Matching Statistics: Per ACE feature is dependent on the QoS Packet Matching Statistics feature. Therefore, the following prerequisites apply:

- If the QoS Packet-Matching Statistics: Per Filter is not enabled and a user tries to enable the QoS Packet-Matching Statistics: Per ACE feature, the command to enable this feature will be rejected by the CLI. An informational message will be displayed to let the user know why the command was rejected.

- If the QoS Packet-Matching Statistics: Per ACE feature is enabled and a user tries to disable this feature, the command to disable this feature will be rejected by the CLI. An informational message will be displayed to let the user know why the command was rejected.

Restrictions for QoS Packet-Matching Statistics Feature

Enabling the Qos: Packet Matching Statistics feature may increase CPU utilization on a scaled configuration. Before enabling the Qos: Packet Matching Statistics feature, weigh the benefits of the statistics information against the increased CPU utilization for your system.

This section provides information about the restrictions pertaining to the QoS Packet-Matching Statistics: Per Filter feature and the QoS Packet-Matching Statistics: Per ACE feature.

The followings are the restrictions for the QoS Packet Matching Statistics feature:

- Enabling the QoS Packet-Matching Statistics: Per Filter feature may increase CPU utilization on a scaled configuration. Before enabling the QoS Packet-Matching Statistics: Per Filter feature, weigh the benefits of the statistics information against the increase in CPU utilization for your system.

- QoS Packet-Matching Statistics: Per Filter is not supported for the match-all class-maps. However, QoS Packet-Matching Statistics: Per ACE is supported for the match-all class-maps.

The following table provides information about the QoS Packet-Matching Statistics: Per ACE scaling limitations:

Table 6: QoS Packet-Matching Statistics: Per ACE Scaling Limitations

Platform	ACEs (IPv4 or IPv6)
ASR1000-ESP5, ASR1001, ASR1002-F, ASR1002-X	25,000
ASR1000-ESP10	30,000
ASR1000-ESP20/ESP40/ESP100	30,000
ISR4400	20,000
CSR1000V	1,000

Information About QoS Packet-Matching Statistics

This section provides an overview of the QoS Packet-Matching Statistics: Per Filter feature and the QoS Packet-Matching Statistics: Per ACE feature.

QoS Packet-Matching Statistics: Per Filter Feature Overview

The QoS Packet-Matching Statistics: Per Filter feature allows you to count and display the number of packets and bytes matching a filter.

To define a filter, use the **class-map** command with the **match-any** keyword, for example:

```
class-map match-any my_class
   match ip precedence 4 <-------- User-defined filter
   match qos-group 10 <-------- User-defined filter
```

Using this information, you can perform the following tasks:

- Compare the amount of voice traffic with the amount of data traffic on a segment of your network
- Adjust bandwidth availability
- Accurately determine billing
- Troubleshoot service problems

The system collects packet matching statistics in 10-second cycles. If there are many interfaces or sessions, the system collects statistics for about 8000 of them during each cycle. In a scaled configuration, several 10-second cycles may be required to gather all the statistics.

QoS Packet-Matching Statistics: Per ACE Feature Overview

The QoS Packet-Matching Statistics: Per ACE feature allows you to track and display the number of packets and bytes matching individual ACEs that are used in QoS policies (access groups used in class maps).

This feature provides hit counters for ACEs used in QoS policies. When this feature is enabled, it will add QoS hit counters for the ACEs used in a QoS policy to the existing security access list counters for that particular ACE. The access list counters can be seen in the following command output:

```
Router# show ip access-lists

Extended IP access list A1
    10 permit ip 32.1.1.0 0.0.0.255 any (129580275 matches)
Extended IP access list A6and7
    10 permit ip 32.1.6.0 0.0.0.255 any (341426749 matches)
    20 permit ip 32.1.7.0 0.0.0.255 any (398245767 matches)
Extended IP access list source
    10 permit ip any host 16.1.1.5 (16147976 matches)
```

The QoS hit counters (for the ACEs used in QoS policies) will be added to the access list counters. We recommend that you pay attention to the following points when you enable this feature:

- Access list counts are not interface specific, as can be seen in the output of the **show ip access-lists** command (there is no mention of interface). They are aggregate counters of all the hits, for all the features that use the ACEs and support the counts, across all interfaces and directions.
- Interface-specific counts are provided in the existing QoS command(**show policy-map interface**) if the QoS Packet-Matching Statistics: Per Filter feature is enabled. However, the command specified previously shows only the counts per filter (ACL or access group), not per ACE, as can be seen in the following sample output:

```
Router# show access-lists

Extended IP access list A1
    10 permit ip 32.1.1.0 0.0.0.255 any (2000 matches)

Router# show policy-map interface GigabitEthernet0/0/2

  Service-policy input: simple

    Class-map: A1-class (match-all)
      1000 packets, 124000 bytes
      5 minute offered rate 4000 bps
      Match: access-group name A1

    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 261000 bps, drop rate 0 bps
      Match: any
```

- If an ACE is present in a QoS filter (match statement within a class map), but the packet does *not* match the statement, the ACE counter will *not* be incremented for that packet. This can happen if:
  - The ACE is used in a deny statement.
  - Other matching criteria in a match-all class map definition (such as match ip prec 1) prevent the packet from matching the class.
  - Other matching criteria in a match-any class map definition (such as match ip prec 1) match the packet and keep it from matching the ACE match criteria. (This filter precedes the ACE filter and the packet matches both the statements).
- Access list counts are an aggregate (for a particular ACE) of the hit counts for all the features using that ACE, and support the per ACE counts. (In Cisco IOS XE3.10, only Security and QoS ACLs support per ACE counts, but that may change in future releases). Therefore, it is possible that a single packet will hit (and be counted by) multiple features using the same ACE and hence result in multiple counts for the same packet (as it traverses each feature). The following is an example of this:

```
ip access-list extended A1
  permit ip 32.1.1.0 0.0.0.255 any
class-map match-all A1-class
  match access-group name A1

interface GigabitEthernet0/0/2
 ip address 32.0.0.1 240.0.0.0
 ip access-group A1 in
 duplex auto
 speed auto
 media-type rj45
 no negotiation auto
 service-policy input simple


Router# show access-lists

Extended IP access list A1
    10 permit ip 32.1.1.0 0.0.0.255 any (2000 matches)

Router# show policy-map interface GigabitEthernet0/0/2

  Service-policy input: simple
```

```
Class-map: A1-class (match-all)
  1000 packets, 124000 bytes
  5 minute offered rate 4000 bps
  Match: access-group name A1

Class-map: class-default (match-any)
  0 packets, 0 bytes
  5 minute offered rate 261000 bps, drop rate 0 bps
  Match: any
```

How to Configure QoS Packet-Matching Statistics

This section provides information about how to configure QoS Packet-Matching Statistics.

Configuring QoS Packet-Matching Statistics: Per Filter

Before you begin

- Before enabling the QoS Packet-Matching Statistics: Per Filter feature, ensure that no policy-maps are associated with the interfaces on the system. If they are, the system returns the following message:

```
Either a) A system RELOAD or
       b) Remove all service-policies, re-apply the change
            to the statistics, re-apply all service-policies
   is required before this command will be activated.
```

- Before enabling the QoS Packet-Matching Statistics: Per Filter feature, ensure that you have defined a filter that is using the **class-map** command with the **match-any** keyword.

Note Enabling the QoS Packet-Matching Statistics: Per Filter feature may increase CPU utilization on a scaled configuration. Before enabling the QoS Packet-Matching Statistics: Per Filter feature, weigh the benefits of the statistics information against an increase in CPU utilization for your system.

To configure the QoS Packet-Matching Statistics: Per Filter feature, perform the following procedure:

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **platform qos match-statistics per-filter**
4. **interface** *interface -name*
5. **service-policy** {**input** | **output**} *policy-map-name*
6. **end**
7. **show policy-map interface** *interface-name*
8. **configure terminal**
9. **interface** *interface-name*

10. **no service-policy** {**input** | **output**} *policy-map-name*
11. **exit**
12. **no platform qos match-statistics per-filter**
13. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Router> enable`	Enables the privileged EXEC mode. Enter your password, if prompted.
Step 2	**configure terminal** **Example:** `Router# configure terminal`	Enters the global configuration mode.
Step 3	**platform qos match-statistics per-filter** **Example:** `Router(config)# platform qos match-statistics per-filter`	Enables the QoS Packet-Matching Statistics: Per Filter feature.
Step 4	**interface** *interface -name* **Example:** `Router(config)# interface GigabitEthernet0/0/0`	Specifies the interface for attaching the policy-map.
Step 5	**service-policy** {**input** | **output**} *policy-map-name* **Example:** `Router(config-if)# service-policy input pol1`	Attaches a QoS policy-map to the interface. The QoS Packet Matching Statistics feature should be enabled before attaching any QoS policies.
Step 6	**end** **Example:** `Router# end`	Exits the configuration mode.
Step 7	**show policy-map interface** *interface-name* **Example:** `Router# show policy-map interface serial4/0/0`	Displays the packet statistics of all the classes that are configured for all the service policies that are present on the specified interface, subinterface, or a specific permanent virtual circuit (PVC) on the interface.
Step 8	**configure terminal** **Example:** `Router# configure terminal`	Enters the global configuration mode.

	Command or Action	Purpose
Step 9	**interface** *interface-name* **Example:** `Router(config)# interface GigabitEthernet0/0/0`	Specifies the interface for removing the policy-map.
Step 10	**no service-policy** {**input** | **output**} *policy-map-name* **Example:** `Router(config-if)# no service-policy input pol1`	Removes a QoS policy-map from an interface. All the QoS policies should be removed from the interfaces before the QoS Packet Matching Statistics feature can be disabled.
Step 11	**exit** **Example:** `Router(config-if)# exit`	Exits the interface configuration mode.
Step 12	**no platform qos match-statistics per-filter** **Example:** `Router(config)# no platform qos match-statistics per-filter`	Disables the QoS Packet-Matching Statistics: Per Filter feature.
Step 13	**end** **Example:** `Router# end`	Exits the configuration mode.

Examples

Use the **show policy-map interface** command to display the packet statistics of all the classes that are configured for all the service policies that are present on the specified interface, subinterface, or a specific PVC on the interface:

```
Router# show policy-map interface gig1/1/0

 GigabitEthernet1/1/0
  Service-policy input: pol1     ! target = gig1/1/0,input
    Class-map: class1 (match-any)
      1000 packets, 40000 bytes
      5 minute offered rate 0000 bps, drop rate 0000 bps
      Match: ip precedence 1 <-------- User-defined filter
        800 packets, 32000 bytes <-------- Filter matching results
      Match: ip precedence 2 <-------- User-defined filter
        200 packets, 8000 bytes <-------- Filter matching results
      QoS Set
        ip precedence 7
          No packet marking statistics available
    Class-map: class-default (match-any)
      500 packets, 20000 bytes
      5 minute offered rate 0000 bps, drop rate 0000 bps
      Match: any <-------- User-defined filter
        500 packets, 20000 bytes <-------- Filter matching results
```

Configuring QoS Packet-Matching Statistics: Per ACE

Before you begin

Before enabling the QoS Packet-Matching Statistics: Per ACE feature, ensure that the QoS Packet-Matching Statistics: Per Filter feature has been enabled.

The following example shows how to check the feature status by using the **show platform hardware qfp active feature qos configuration global** command:

```
Router# show platform hardware qfp active feature qos configuration global
Marker statistics are: disabled
Match per-filter statistics are: enabled <<<<<<<
Match per-ace statistics are: enabled <<<<<<
Performance-Monitor statistics are: disabled
```

To configure the QoS Packet-Matching Statistics: Per ACE feature, perform the following procedure:

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **platform qos match-statistics per-filter**
4. **platform qos match-statistics per-ace**
5. **interface** *interface-name*
6. **service-policy**{**input**|**output**}*policy-map-name*
7. **end**
8. **show policy-map interface** *interface-name*
9. **show access-lists**
10. **configure terminal**
11. **interface** *interface-name*
12. **no service-policy**{**input**|**output**}*policy-map-name*
13. **exit**
14. **no platform qos match-stat per-ace**
15. **no platform qos match-statistics per-filter**
16. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Router> enable`	Enables the privileged EXEC mode. Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Router# configure terminal`	Enters the global configuration mode.

	Command or Action	Purpose
Step 3	**platform qos match-statistics per-filter** **Example:** `Router(config)# platform qos match-statistics per-filter`	Enables the QoS Packet-Matching Statistics: Per Filter feature.
Step 4	**platform qos match-statistics per-ace** **Example:** `Router(config)# platform qos match-statistics per-ace`	Enables the QoS Packet-Matching Statistics: Per ACE feature.
Step 5	**interface** *interface-name* **Example:** `Router(config)# interface GigabitEthernet0/0/0`	Specifies the interface for attaching the policy-map.
Step 6	**service-policy** {**input**|**output**} *policy-map-name* **Example:** `Router(config-if)# service-policy input pol1`	Attaches a QoS policy-map to an interface. The QoS Matching Statistics feature should be enabled before attaching QoS policies.
Step 7	**end** **Example:** `Router# end`	Exits the configuration mode.
Step 8	**show policy-map interface** *interface-name* **Example:** `Router# show policy-map interface serial4/0/0`	Displays the packet statistics pertaining to all the classes that are configured for all the service policies either on the specified interface, subinterface, or on a specific PVC on the interface.
Step 9	**show access-lists** **Example:** `Router# show access-lists`	Displays the contents of current access lists, including the QoS Packet-Matching Statistics: Per ACE.
Step 10	**configure terminal** **Example:** `Router# configure terminal`	Enters the global configuration mode.
Step 11	**interface** *interface-name* **Example:** `Router(config)# interface GigabitEthernet0/0/0`	Specifies the interface for removing the policy-map.
Step 12	**no service-policy** {**input**|**output**} *policy-map-name* **Example:** `Router(config-if)# no service-policy input pol1`	Removes a QoS policy-map from an interface. All the QoS policies should be removed from the interfaces before the QoS Matching Statistics feature can be disabled.

	Command or Action	Purpose
Step 13	**exit** **Example:** `Router(config-if)# exit`	Exits the interface configuration mode.
Step 14	**no platform qos match-stat per-ace** **Example:** `Router(config)# no platform qos match-stat per-ace`	Disables the QoS Packet-Matching Statistics: Per ACE feature.
Step 15	**no platform qos match-statistics per-filter** **Example:** `Router(config)# no platform qos match-statistics per-filter`	Disables the QoS Packet-Matching Statistics: Per Filter feature.
Step 16	**end** **Example:** `Router# end`	Exits the configuration mode.

Example

Use the **show policy-map interface** command to display the per-filter statistics of all the classes that are configured for all the service policies on the specified interface, subinterface, or on a specific PVC on the interface:

```
Router# show policy-map interface GigabitEthernet0/0/2


  Service-policy input: test-match-types

    Class-map: A1orA2-class (match-any)
      482103366 packets, 59780817384 bytes
      5 minute offered rate 6702000 bps
      Match: access-group name A1
        62125633 packets, 7703578368 bytes
        5 minute rate 837000 bps
      Match: access-group name A2
        419977732 packets, 52077238892 bytes
        5 minute rate 5865000 bps

    Class-map: A3andprec1-class (match-all)
      5673520 packets, 703516480 bytes
      5 minute offered rate 837000 bps
      Match: access-group name A3
      Match: ip precedence 1

    Class-map: A5-class (match-all)
      227101820 packets, 28160625680 bytes
      5 minute offered rate 3351000 bps
      Match: access-group name A5
```

```
    Class-map: A6and7-class (match-all)
      627615840 packets, 77824340228 bytes
      5 minute offered rate 9215000 bps
      Match: access-group name A6and7

    Class-map: A3-class (match-all)
      111548288 packets, 13831987712 bytes
      5 minute offered rate 1675000 bps
      Match: access-group name A3

    Class-map: A4andsource (match-all)
      16115590 packets, 1998333160 bytes
      5 minute offered rate 2513000 bps
      Match: access-group name A4
      Match: access-group name source

    Class-map: class-default (match-any)
      164881212 packets, 20445270288 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
```

Use the **show ip access-lists** command to display the contents of current access lists (which includes the QoS Packet-Matching Statistics: Per ACE):

```
Router# show ip access-lists

Extended IP access list A1
    10 permit ip 32.1.1.0 0.0.0.255 any (129580275 matches)
Extended IP access list A2
    10 permit ip 32.1.2.0 0.0.0.255 any (486342300 matches)
Extended IP access list A3
    10 permit ip 32.1.3.0 0.0.0.255 any (306738457 matches)
Extended IP access list A4
    10 permit ip 32.1.4.0 0.0.0.255 any (16147975 matches)
Extended IP access list A5
    10 permit ip 32.1.5.0 0.0.0.255 any (294357455 matches)
Extended IP access list A6and7
    10 permit ip 32.1.6.0 0.0.0.255 any (341426749 matches)
    20 permit ip 32.1.7.0 0.0.0.255 any (398245767 matches)
Extended IP access list source
    10 permit ip any host 16.1.1.5 (16147976 matches)
```

Troubleshooting Tips

To confirm that the QoS: Packet Matching Statistics feature is enabled, use the **show platform hardware qfp active feature qos config global** command. If the feature is disabled, you should see a message similar to the following:

Router# **show platform hardware qfp active feature qos config global**

```
Marker statistics are: enabled
Match per filter statistics are: enabled
```

Example: Configuring a QoS Packet-Matching Statistics: Per Filter

The following example shows how to configure a QoS Packet-Matching Statistics: Per Filter, perform the following tasks:

- Define a QoS packet matching filter
- Display the **show policy-map interface** command output

```
Router# show policy-map interface Tunnel1

  Service-policy output: DATA-OUT-PARENT
    Class-map: class-default (match-any)
      4469 packets, 4495814 bytes
      5 minute offered rate 0000 bps, drop rate 0000 bps
      Match: any <-------- User-defined filter
      Queueing
      queue limit 416 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 4469/4558380
      shape (average) cir 100000000, bc 400000, be 400000
      target shape rate 100000000
      Service-policy : DATA-OUT
        queue stats for all priority classes:
          Queueing
          queue limit 200 packets
          (queue depth/total drops/no-buffer drops) 0/0/0
          (pkts output/bytes output) 4469/4558380
        Class-map: ATM-VTI-RIP-SPK1-DATA (match-any)
          4469 packets, 4495814 bytes <-------- Filter matching results
          5 minute offered rate 0000 bps, drop rate 0000 bps
          Match: access-group 121 <-------- User-defined filter
            4469 packets, 4495814 bytes <-------- Filter matching results
            5 minute rate 0 bps
          QoS Set
            ip precedence 3
              Packets marked 4469
          Priority: 100 kbps, burst bytes 2500, b/w exceed drops: 0
```

Additional References

Related Documents

Related Topic	Document Title
Cisco IOS commands	Cisco IOS Master Commands List, All Releases
Quality of service commands	*Cisco IOS Quality of Service Command Reference*

Standards

Standard	Title
No new or modified standards are supported, and support for existing standards has not been modified.	--

MIBs

MIB	MIBs Link
CISCO-CLASS-BASED-QOS-MIB	To locate and download MIBs for selected platforms, Cisco software releases, and feature sets, use Cisco MIB Locator found at the following URL: http://www.cisco.com/go/mibs

RFCs

RFC	Title
No new or modified RFCs are supported, and support for existing RFCs has not been modified.	--

Technical Assistance

Description	Link
The Cisco Support and Documentation website provides online resources to download documentation, software, and tools. Use these resources to install and configure the software and to troubleshoot and resolve technical issues with Cisco products and technologies. Access to most tools on the Cisco Support and Documentation website requires a Cisco.com user ID and password.	http://www.cisco.com/cisco/web/support/index.html

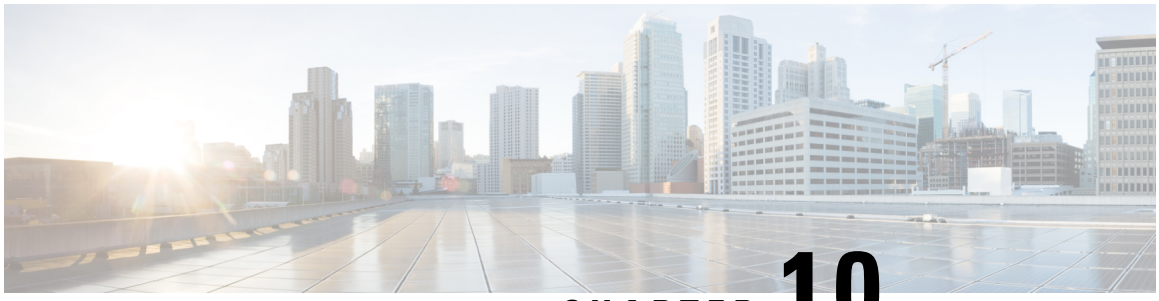
Feature Information for QoS Packet-Matching Statistics

The following table provides release information about the feature or features described in this module. This table lists only the software release that introduced support for a given feature in a given software release train. Unless noted otherwise, subsequent releases of that software release train also support that feature.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to www.cisco.com/go/cfn. An account on Cisco.com is not required.

Table 7: Feature Information for QoS Packet-Matching Statistics

Feature Name	Releases	Feature Information
QoS Packet-Matching Statistics: Per Filter	Cisco IOS XE Release 3.3S	The QoS Packet-Matching Statistics: Per Filter feature allows you to count and display the number of packets matching individual filters (match statements) used in class-maps within QoS service policies that have. The following commands were introduced or modified: • **platform qos match-statistics per-filter** • **no platform qos match-statistics per-filter** • **show platform hardware qfp active feature qos config global**
QoS Packet-Matching Statistics: Per ACE	Cisco IOS XE Release 3.10S	The QoS Packet-Matching Statistics: Per ACE feature allows you to track and display the number of packets and bytes matching individual ACEs that are used in QoS policies (access groups used in class maps). The following command was introduced: **platform qos match-statistics per-ace**



CHAPTER 10

Set ATM CLP Bit Using Policer

The Set ATM CLP Bit Using Policer feature allows you to police and then mark outbound PPP over ATM (PPPoA) traffic. You can set the ATM cell loss priority (CLP) bit using either of the following methods:

- A policed threshold
- Matching a class

Finding Feature Information

Your software release may not support all the features documented in this module. For the latest caveats and feature information, see Bug Search Tool and the release notes for your platform and software release. To find information about the features documented in this module, and to see a list of the releases in which each feature is supported, see the feature information table.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to https://cfnng.cisco.com/. An account on Cisco.com is not required.

Prerequisites for Set ATM CLP Bit Using Policer

If you are setting the ATM CLP bit by a policed threshold, ensure that a policy-map includes the **set-clp-transmit**action. The new policer action conditionally marks PPPoA traffic in the matched class for a higher drop probability in the ATM network when traffic exceeds a given rate.

If you are setting the ATM CLP bit strictly by matching a class, ensure that a policy-map includes the **set atm-clp** action. The set directive marks all traffic in the matched class for higher drop probability in the ATM network.

You can attach policy-maps with the **set-clp-transmit**or **set atm-clp**actions to a virtual template. This template is cloned when PPPoA sessions are created or by dynamic assignment.

Information About Set ATM CLP Bit Using Policer

ATM CLP Bit

The ATM CLP bit shows the drop priority of the ATM cell. During ATM network congestion, the router discards ATM cells with the CLP bit set to 1 before discarding cells with a CLP bit setting of 0.

Using the Set ATM CLP Bit Using Policer feature, you can configure the **police** command to enable the ATM CLP bit in cell headers. The ATM CLP bit can be explicitly marked by a set directive.

The Set ATM CLP Bit Using Policer feature supports the **set-clp-transmit** policing action in the following types of policies:

- Single-rate policing
- Dual-rate policing
- Hierarchical

How to Set the ATM CLP Bit Using Policer

Configuring PPPoA Broadband Traffic Policing

Before you begin

Before configuring the policy-map, ensure that you have defined any class maps used to classify traffic.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **policy-map** *policy-map-name*
4. **class** {*class-name*| **class-default**]
5. **police** [**cir** *cir*] [**conform-action** *action*] [**exceed-action** *action*]
6. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.

	Command or Action	Purpose
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**policy-map** *policy-map-name* **Example:** `Device(config)# policy-map parent-policy`	Enters policy-map configuration mode and creates a policy-map.
Step 4	**class** {*class-name*| **class-default**] **Example:** `Device(config-pmap)# class class-default`	Enters policy-map class configuration mode. Specifies the name of the class whose policy you want to create or change or specifies the default class (commonly known as the class-default class) before you configure its policy. Repeat this command as many times as necessary to specify the child or parent classes that you are creating or modifying: • *class name* --Name of the class to be configured or whose policy is to be modified. The class name is used for both the class map and to configure a policy for the class in the policy-map. • **class-default** --Specifies the default class so that you can configure or modify its policy.

Step 5	**police** [**cir** *cir*] [**conform-action** *action*] [**exceed-action** *action*]
	Example:
	`Device(config-pmap-c)# police 1000000`
	Example:
	`Router(config-pmap-c-police)# conform-action`
	Example:
	`transmit`
	Example:
	`Device(config-pmap-c-police)# exceed-action`
	Example:
	`set-clp-transmit`
		Configures traffic policing and specifies multiple actions applied to packets marked as conforming to, exceeding, or violating a specific rate.
		• Enters policy-map class police configuration mode. Use one line per action that you want to specify:
		  • **cir**--(Optional) Committed information rate. Indicates that the CIR will be used for policing traffic.
		  • **conform-action**--(Optional) Action to take on packets when the rate is less than the conform burst.
		  • **exceed-action**--(Optional) Action to take on packets whose rate is within the conform and conform plus exceed burst.

Step 6	**end** **Example:**	(Optional) Returns to privileged EXEC mode.

	Command or Action	Purpose
	`Device(config-pmap-c)# end`	

Example

The following example shows you how to set the ATM CLP using a policer:

```
policy-map egress_atm_clp_policer
 class prec0
  police cir 5000000
 class prec1
  police cir 3000000 conform-action transmit exceed-action set-clp-transmit
 class class-default
  police cir 1000000 conform-action transmit exceed-action set-clp-transmit
```

Marking the ATM CLP Bit

Before you begin

Before configuring the policy-map, ensure that you have defined any class maps used to classify traffic.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **policy-map policy-map-name**
4. **class** {*class-name*| **class-default**]
5. **set atm-clp**
6. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Router> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Router# configure terminal`	Enters global configuration mode.
Step 3	**policy-map policy-map-name** **Example:** `Router(config)# policy-map parent-policy`	Enters policy-map configuration mode and creates a policy-map.

	Command or Action	Purpose
Step 4	**class** {*class-name*| **class-default**] **Example:** `Router(config-pmap)# class class-default`	Enters policy-map class configuration mode. Specifies the name of the class whose policy you want to create or change or specifies the default class (commonly known as the class-default class) before you configure its policy. Repeat this command as many times as necessary to specify the child or parent classes that you are creating or modifying: • *class name* --Name of the class to be configured or whose policy is to be modified. The class name is used for both the class map and to configure a policy for the class in the policy-map. • **class-default** --Specifies the default class so that you can configure or modify its policy.
Step 5	**set atm-clp** **Example:** `Router(config-pmap-c)# set atm-clp`	Configures marking of the ATM CLP bit for all traffic matching this class.
Step 6	**end** **Example:** `Router(config-pmap-c)# end`	(Optional) Returns to privileged EXEC mode.

Example

The following example shows you how to set the ATM CLP using explicit marking:

```
policy-map egress_atm_clp_policer
 class prec0
  police cir 5000000
 class class-default
  set atm-clp
```

Configuration Examples for Set ATM CLP Bit Using Policer

Example Marking the ATM CLP by Policer Action Matching a Class

This example shows how to do the following:

- Define traffic classes.
- Configure a two-layer policy-map.
- Apply the policy-map to PPPoA sessions.

This policy conditionally marks the ATM CLP bit on the traffic in the matching low_interest class once traffic on the class exceeds a given rate.

```
class-map voice
  match precedence 4
!
class-map web
  match precedence 3
!
class low_interest
  match precedence 1 0
!
policy-map child
 child class voice
  police cir 256000
  priority level 1
 class web
  bandwidth remaining ratio 10
 class low_interest
  police cir 1000000 conform-action transmit exceed-action set-clp-transmit
 class class-default
  bandwidth remaining ratio 1
!
policy-map parent
 class class-default
  shape average 15000000
  service-policy child
```

Policy-maps attached to virtual templates are cloned and used to create a virtual access interface for each PPPoA session:

```
interface Virtual-Template1
 ip unnumbered Loopback1
 load-interval 30
 peer default ip address pool POOL1
 ppp authentication chap ppp
 ipcp address required
 service-policy output parent
```

Example Marking the ATM CLP by Policer Action Policed Threshold

This example shows how to do the following:

- Define traffic classes.
- Configure a two-layer policy-map.
- Apply the policy-map to PPPoA sessions.

This policy marks all non-essential traffic with the ATM CLP bit so that it is eligible for dropping if the ATM network becomes congested.

```
class-map video
  match precedence 5
!
class-map voice
  match precedence 4
!
class-map web
```

```
  match precedence 3
!
policy-map child
 child class voice
  police cir 256000
  priority level 1
 class video
  police cir 4000000
  priority level 2
 class web
  set atm-clp
  bandwidth remaining ratio 10
 class class-default
  bandwidth remaining ratio 1
  set atm-clp
!
interface Virtual-Template1
 ip unnumbered Loopback1
 load-interval 30
 peer default ip address pool POOL1
 ppp authentication chap ppp
 ipcp address required
 service-policy output parent
```

Additional References

Related Documents

Related Topic	Document Title
Cisco IOS commands	Cisco IOS Master Commands List, All Releases
Quality of Service commands	*Cisco IOS Quality of Service Command Reference*

Standards

Standard	Title
No new or modified standards are supported by this feature, and support for existing standards has not been modified by this feature.	--

MIBs

MIB	MIBs Link
No new or modified MIBs are supported by this feature, and support for existing MIBs has not been modified by this feature.	To locate and download MIBs for selected platforms, Cisco software releases, and feature sets, use Cisco MIB Locator found at the following URL: http://www.cisco.com/go/mibs

RFCs

RFC	Title
No new or modified RFCs are supported by this feature, and support for existing RFCs has not been modified by this feature.	--

Technical Assistance

Description	Link
The Cisco Support and Documentation website provides online resources to download documentation, software, and tools. Use these resources to install and configure the software and to troubleshoot and resolve technical issues with Cisco products and technologies. Access to most tools on the Cisco Support and Documentation website requires a Cisco.com user ID and password.	http://www.cisco.com/cisco/web/support/index.html

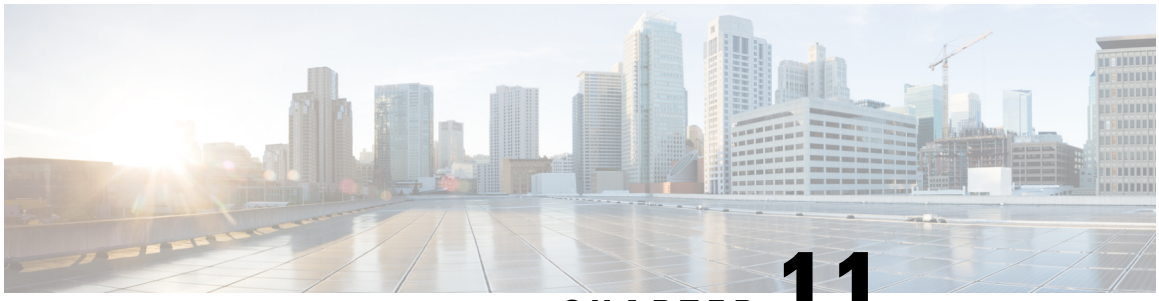
Feature Information for Set ATM CLP Bit Using Policer

The following table provides release information about the feature or features described in this module. This table lists only the software release that introduced support for a given feature in a given software release train. Unless noted otherwise, subsequent releases of that software release train also support that feature.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to www.cisco.com/go/cfn. An account on Cisco.com is not required.

Table 8: Feature Information for Set ATM CLP Bit Using Policer

Feature Name	Releases	Feature Information
Set ATM CLP Bit Using Policer	Cisco IOS Release XE 3.3S Cisco IOS Release XE 3.14S	The Set ATM CLP Bit Using Policer feature allows you to police and then mark outbound PPPoA traffic. In Cisco IOS Release XE 3.14S, support for this feature was added on the Cisco 4451-X Integrated Services Router. The following commands were introduced or modified: **set atm-clp**and **police**.



CHAPTER 11

EVC Quality of Service

This document contains information about how to enable quality of service (QoS) features (such as traffic classification and traffic policing) for use on an Ethernet virtual circuit (EVC).

An EVC as defined by the Metro Ethernet Forum is a port-level point-to-point or multipoint-to-multipoint circuit. It is an end-to-end representation of a single instance of a service being offered by a provider to a customer. It embodies the different parameters on which the service is being offered.

Finding Feature Information

Your software release may not support all the features documented in this module. For the latest caveats and feature information, see Bug Search Tool and the release notes for your platform and software release. To find information about the features documented in this module, and to see a list of the releases in which each feature is supported, see the feature information table.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to https://cfnng.cisco.com/. An account on Cisco.com is not required.

Information About Quality of Service on an EVC

EVC Quality of Service and the MQC

QoS functionality is typically applied using traffic classes, class maps, and policy-maps. For example, you can specify that traffic belonging to a particular class be grouped into specific categories, and receive a specific QoS treatment (such as classification or policing). The QoS treatment the traffic is to receive is specified in a policy-map and the policy-map is attached to an interface. The mechanism used for applying QoS in this manner is the modular QoS CLI (MQC.)

The policy-map can be attached to an interface in either the incoming (ingress) or outgoing (egress) direction with the **service-policy** command.

The MQC structure allows you to define a traffic class, create a traffic policy, and attach the traffic policy to an interface (in this case, an EVC).

The MQC structure consists of the following three high-level steps.

1. Define a traffic class by using the **class-map**command. A traffic class is used to classify traffic.
2. Create a traffic policy by using the **policy-map** command. (The terms *traffic policy* and *policy-map* are often synonymous.) A traffic policy (policy-map) contains a traffic class and one or more QoS features that will be applied to the traffic class. The QoS features in the traffic policy determine how to treat the classified traffic.
3. Attach the traffic policy (policy-map) to the interface by using the **service-policy** command.

Note For more information about the MQC, including information about hierarchical policy-maps and class maps, see the "Applying QoS Features Using the MQC" module.

QoS-Aware Ethernet Flow Point (EFP)

As described in the EVC Quality of Service and the MQC, on page 189, the MQC is used to apply one or more QoS features to network traffic. The last step in using the MQC is to attach the traffic policy (policy-map) to an interface (in this case, an EVC) by using the **service-policy** command.

With the EVC Quality of Service feature, the **service-policy** command can be used to attach the policy-map to an Ethernet Flow Point (EFP) in either the incoming (ingress) *or* outgoing (egress) direction of an EVC. This way, the EFP is considered to be "QoS-aware."

QoS Functionality and EVCs

The specific QoS functionality includes the following:

- Packet classification (for example, based on differentiated services code point (DSCP) value and QoS group identifier)
- Packet marking (for example, based on Class of Service (CoS) value)
- Traffic policing (two- and three-color and multiple actions)
- Bandwidth sharing
- Priority queueing (in the outbound direction on the EVC only)
- Weighted Random Early Detection (WRED)

The QoS functionality is enabled by using the appropriate commands listed in the following sections.

match Commands Supported by EVC QoS for Classifying Traffic

The table below lists *some* of the available **match** commands that can be used when classifying traffic on an EVC. The available **match** commands vary by Cisco IOS XE release. For more information about the commands and command syntax, see the Cisco IOS Quality of Service Solutions Command Reference.

Table 9: match Commands That Can Be Used with the MQC

Command	Purpose
match access-group	Configures the match criteria for a class map on the basis of the specified access control list (ACL).
match any	Configures the match criteria for all packets.
match cos	Matches a packet based on a Layer 2 CoS marking.
match cos inner	Matches the inner CoS of QinQ packets on a Layer 2 CoS marking.
match [**ip**] **dscp**	Identifies a specific IP DSCP value as a match criterion. Up to eight DSCP values can be included in one match statement.
match not	Specifies the single match criterion value to use as an unsuccessful match criterion. **Note** The **match not** command, rather than identifying the specific match parameter to use as a match criterion, is used to specify a match criterion that prevents a packet from being classified as a member of the class. For instance, if the **match not qos-group 6**command is issued while you configure the traffic class, QoS group 6 becomes the only QoS group value that is not considered a successful match criterion. All other QoS group values would be successful match criteria.
match [**ip**] **precedence**	Identifies IP precedence values as match criteria.
match qos-group	Identifies a specific QoS group value as a match criterion.
match source-address mac	Uses the source MAC address as a match criterion. **Note** Classifying traffic using the **match source-address mac**command is supported in the input direction only.
match vlan (QoS)	Matches and classifies traffic on the basis of the VLAN identification number.
match vlan inner	Configures a class map to match the innermost VLAN ID in an 802.1q tagged frame.

Multiple match Commands in One Traffic Class

If the traffic class contains more than one **match** command, you need to specify how to evaluate the **match** commands. You specify this by using either the **match-any** or **match-all**keyword of the **class-map** command. Note the following points about the **match-any** and **match-all** keywords:

- If you specify the **match-any**keyword, the traffic being evaluated by the traffic class must match *one* of the specified criteria.
- If you specify the **match-all** keyword, the traffic being evaluated by the traffic class must match *all* of the specified criteria.
- If you do not specify either keyword, the traffic being evaluated by the traffic class must match *all* of the specified criteria (that is, the behavior of the **match-all** keyword is used).

Commands Used to Enable QoS Features on the EVC

The commands used to enable QoS features vary by Cisco IOS XE release. The table below lists *some* of the available commands and the QoS features that they enable. For complete command syntax, see the Cisco IOS Quality of Service Solutions Command Reference.

For more information about a specific QoS feature that you want to enable, see the appropriate module of the Cisco IOS Quality of Service Solutions Configuration Guide.

Table 10: Commands Used to Enable QoS Features

Command	Purpose
bandwidth	Configures a minimum bandwidth guarantee for a class.
bandwidth remaining	Configures an excess weight for a class.
drop	Discards the packets in the specified traffic class.
fair-queue	Enables the flow-based queueing feature within a traffic class.
police	Configures traffic policing. Allows specifying of multiple policing actions.
police (percent)	Configures traffic policing on the basis of a percentage of bandwidth available on an interface.
police (two rates)	Configures traffic policing using two rates, the committed information rate (CIR) and the peak information rate (PIR).
priority	Gives priority to a class of traffic belonging to a policy-map.
queue-limit	Specifies or modifies the maximum number of packets the queue can hold for a class configured in a policy-map.
random-detect	Enables Weighted Random Early Detection (WRED).
random-detect cos-based	Enables Weighted random early detection (WRED) on the basis of the class of service (CoS) value of a packet.
random-detect dscp-based	Specifies that Weighted random early detection (WRED) is to use the differentiated services code point (DSCP) value when it calculates the drop probability for a packet.
random-detect discard-class	Configures the WRED parameters for a discard-class value for a class in a policy-map.

Command	Purpose
random-detect discard-class-based	Configures WRED on the basis of the discard class value of a packet.
random-detect exponential-weighting-constant	Configures the exponential weight factor for the average queue size calculation for the queue reserved for a class.
random-detect precedence	Configure the WRED parameters for a particular IP Precedence for a class policy in a policy-map.
service-policy	Specifies the name of a traffic policy used as a matching criterion (for nesting traffic policies [hierarchical traffic policies] within one another).
set cos	Sets the Layer 2 CoS value of an outgoing packet.
set cos-inner	Marks the inner class of service field in a bridged frame.
set discard-class	Marks a packet with a discard-class value.
set [**ip**] **dscp**	Marks a packet by setting the DSCP value in the type of service (ToS) byte.
set mpls experimental	Designates the value to which the Multiprotocol Label Switching (MPLS) bits are set if the packets match the specified policy-map.
set precedence	Sets the precedence value in the packet header.
set qos-group	Sets a QoS group identifier (ID) that can be used later to classify packets.
shape	Shapes traffic to the indicated bit rate according to the algorithm specified.

input and output Keywords of the service-policy Command

As a general rule, the QoS features configured in the traffic policy can be applied to packets entering the interface or to packets leaving the interface. Therefore, when you use the **service-policy** command, you need to specify the direction of the traffic policy by using the **input** or **output** keyword.

For instance, the **service-policy output policy-map1** command would apply the QoS features in the traffic policy to the interface in the output direction. All packets leaving the interface (output) are evaluated according to the criteria specified in the traffic policy named policy-map1.

Note For Cisco releases, queueing mechanisms are not supported in the input direction. Nonqueueing mechanisms (such as traffic policing and traffic marking) are supported in the input direction. Also, classifying traffic on the basis of the source MAC address (using the **match source-address mac** command) is supported in the input direction only.

How to Configure a Quality of Service Feature on an EVC

Creating a Traffic Class for Use on the EVC

To create a traffic class, use the **class-map** command to specify the traffic class name. Then use one or more **match** commands to specify the appropriate match criteria. Packets matching the criteria that you specify are placed in the traffic class.

To create the traffic class for use on the EVC, complete the following steps.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **class-map** [**match-all** | **match-any**] *class-name*
4. **match cos** *cos-number*
5. Enter additional **match** commands, if applicable; otherwise, proceed with the next step.
6. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Router> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Router# configure terminal`	Enters global configuration mode.
Step 3	**class-map** [**match-all**	**match-any**] *class-name* **Example:** `Router(config)# class-map match-any class1`	Creates a class map and enters class-map configuration mode. • The class map is used for matching packets to the specified class. **Note** The **match-all** keyword specifies that all match criteria must be met. The **match-any** keyword specifies that one of the match criteria must be met. Use these keywords only if you will be specifying more than one **match** command.
Step 4	**match cos** *cos-number* **Example:**	Matches a packet on the basis of a Layer 2 CoS number. **Note** The **match cos** command is an example of a **match** command you can use.

	Command or Action	Purpose
	`Router(config-cmap)# match cos 2`	
Step 5	Enter additional **match** commands, if applicable; otherwise, proceed with the next step.	--
Step 6	**end** **Example:** `Router(config-cmap)# end`	(Optional) Exits class map configuration mode and returns to privileged EXEC mode.

Creating a Policy-Map for Use on the EVC

To create a traffic policy (or policy-map) for use on the EVC, complete the following steps.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **policy-map** *policy-map-name*
4. **class** {*class-name*| **class-default**}
5. **police** *bps* [*burst-normal*] [*burst-max*] [**conform-action** *action*] [**exceed-action** *action*] [**violate-action** *action*]
6. Enter the commands for any additional QoS feature that you want to enable, if applicable; otherwise, proceed to the next step.
7. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Router> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Router# configure terminal`	Enters global configuration mode.

Step 3	**policy-map** *policy-map-name*
	Example:
	`Router(config)#`
	`policy-map policy1`
		Creates or specifies the name of the traffic policy and enters QoS policy-map configuration mode.

	Command or Action	Purpose

Step 4	**class** {*class-name*| **class-default**}
	Example:
	`Router(config-pmap)#`
	`class class1`
		Specifies the name of a class and enters QoS policy-map class configuration mode.
		Note This step associates the traffic class with the traffic policy.

Step 5	**police** *bps* [*burst-normal*] [*burst-max*] [**conform-action** *action*] [**exceed-action** *action*] [**violate-action** *action*] **Example:** `Router(config-pmap-c)# police 3000`	(Optional) Configures traffic policing. **Note** The **police**command is an example of a command that you can use in a policy-map to enable a QoS feature.
Step 6	Enter the commands for any additional QoS feature that you want to enable, if applicable; otherwise, proceed to the next step.	--
Step 7	**end** **Example:** `Router(config-pmap-c)# end`	(Optional) Exits QoS policy-map class configuration mode and returns to privileged EXEC mode.

Configuring the EVC and Attaching a Traffic Policy to the EVC

The traffic policy (policy-map) applies the enabled QoS feature to the traffic class once you attach the policy-map to the EVC.

To configure the EVC and attach a traffic policy to the EVC, complete the following steps.

Note One of the commands used to attach the traffic policy to the EVC is the **service-policy** command. When you use this command, you must specify either the **input** or **output** keyword along with the policy-map name. The policy-map contains the QoS feature you want to use. Certain QoS features can only be used in either the input or output direction. For more information about these keywords and the QoS features supported, see the input and output Keywords of the service-policy Command, on page 193. Also, if you attach a traffic policy to an interface containing multiple EVCs, the traffic policy will be attached to *all* of the EVCs on the interface.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **interface** *interface-type interface-number*
4. **service instance** *id* **ethernet** [*evc-name*]
5. **encapsulation dot1q** *vlan-id* [,*vlan-id*[-*vlan-id*]] [**native**]
6. **rewrite ingress tag translate 1-to-1 dot1q** *vlan-id* **symmetric**
7. **bridge domain** *domain-number*

8. **service-policy** {**input** | **output**} *policy-map-name*
9. **end**
10. **show policy-map interface** *type number* **service instance** *service-instance-number*

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Router> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Router# configure terminal`	Enters global configuration mode.
Step 3	**interface** *interface-type interface-number* **Example:** `Router(config)# interface gigabitethernet 0/0/1`	Configures an interface type and enters interface configuration mode. • Enter the interface type and interface number.
Step 4	**service instance** *id* **ethernet** [*evc-name*] **Example:** `Router(config-if)# service instance 333 ethernet evc1`	Configures an Ethernet service instance on an interface and enters Ethernet service configuration mode. • Enter the service instance identification number and, if applicable, the EVC name (optional).
Step 5	**encapsulation dot1q** *vlan-id* [,*vlan-id*[-*vlan-id*]] [**native**] **Example:** `Router(config-if-srv)# encapsulation dot1q 10`	Defines the matching criteria to map 802.1Q frames ingress on an interface to the appropriate service instance.
Step 6	**rewrite ingress tag translate 1-to-1 dot1q** *vlan-id* **symmetric** **Example:** `Router(config-if-srv)# rewrite ingress tag translate 1-to-1 dot1q 300 symmetric`	Specifies the encapsulation adjustment to be performed on a frame ingressing a service instance.
Step 7	**bridge domain** *domain-number* **Example:** `Router(config-if-srv)# bridge domain 1`	Configures a bridge domain. • Enter the bridge domain number.
Step 8	**service-policy** {**input**	**output**} *policy-map-name* **Example:** `Router(config-if-srv)#`	Attaches a policy-map to an interface. • Enter either the **input** or **output** keyword and the policy-map name.

	Command or Action	Purpose
	`service-policy input policy1`	
Step 9	**end** **Example:** `Router(config-if-srv)# end`	(Optional) Returns to privileged EXEC mode.
Step 10	**show policy-map interface** *type number* **service instance** *service-instance-number* **Example:** `Router# show policy-map interface gigabitethernet 1/0/0 service instance 30`	(Optional) Displays the statistics and the configurations of the input and output policies that are attached to an interface. • Enter the interface type, interface number, and service instance number.

Configuration Examples for EVC Quality of Service

Example Creating a Traffic Class for Use on the EVC

In this example, traffic with a CoS value of 2 is placed in the traffic class called class1:

```
Router> enable
Router# configure terminal
Router(config)# class-map match-any class1
Router(config-cmap)# match cos 2
Router(config-cmap)# end
```

Example Creating a Policy-Map for Use on the EVC

In this example, traffic policing has been configured in the policy-map called policy1. Traffic policing is the QoS feature applied to the traffic in class1:

```
Router> enable
Router# configure terminal
Router(config)#
 policy-map policy1
```

```
Router(config-pmap)#
 class class1

Router(config-pmap-c)# police 3000

Router(config-pmap-c)# end
```

Example Configuring the EVC and Attaching a Traffic Policy to the EVC

In this example, an EVC has been configured and a traffic policy called policy1 has been attached to the EVC:

```
Router> enable

Router# configure terminal

Router(config)# interface gigabitethernet 0/0/1

Router(config-if)# service instance 333 ethernet evc1

Router(config-if-srv)# encapsulation dot1q 10

Router(config-if-srv)# rewrite ingress tag translate 1-to-1 dot1q 300 symmetric

Router(config-if-srv)# bridge domain 1

Router(config-if-srv)# service-policy input policy1

Router(config-if-srv)# end
```

Example Verifying the Traffic Class and Traffic Policy Information for the EVC

The following is sample output of the **show policy-map interface service instance**command. It displays the QoS features configured for and attached to the EFP on the GigabitEthernet interface 1/1/7.

```
Router# show policy-map interface gigabitethernet 1/1/7 service instance 10
 GigabitEthernet1/1/7: EFP 10
  Service-policy input: multiaction
    Class-map: c1 (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0000 bps, drop rate 0000 bps
      Match: ip precedence 3
      police:
          cir 300000 bps, bc 2000 bytes
        conformed 0 packets, 0 bytes; actions:
          set-prec-transmit 7
          set-qos-transmit 10
```

```
    exceeded 0 packets, 0 bytes; actions:
      drop
    conformed 0000 bps, exceed 0000 bps
Class-map: class-default (match-any)
  0 packets, 0 bytes
  5 minute offered rate 0000 bps, drop rate 0000 bps
  Match: any
```

Additional References

Related Documents

Related Topic	Document Title
Cisco IOS commands	Cisco IOS Master Commands List, All Releases
QoS commands: complete command syntax, command modes, command history, defaults, usage guidelines, and examples	*Cisco IOS Quality of Service Solutions Command Reference*
Packet classification	"Classifying Network Traffic" module
Selective Packet Discard	“IPv6 Selective Packet Discard” module

Standards

Standard	Title
No new or modified standards are supported, and support for existing standards has not been modified.	--

MIBs

MIB	MIBs Link
No new or modified MIBs are supported, and support for existing MIBs has not been modified.	To locate and download MIBs for selected platforms, Cisco IOS releases, and feature sets, use Cisco MIB Locator found at the following URL: http://www.cisco.com/go/mibs

RFCs

RFC	Title
No new or modified RFCs are supported, and support for existing RFCs has not been modified.	--

Technical Assistance

Description	Link
The Cisco Support and Documentation website provides online resources to download documentation, software, and tools. Use these resources to install and configure the software and to troubleshoot and resolve technical issues with Cisco products and technologies. Access to most tools on the Cisco Support and Documentation website requires a Cisco.com user ID and password.	http://www.cisco.com/cisco/web/support/index.html

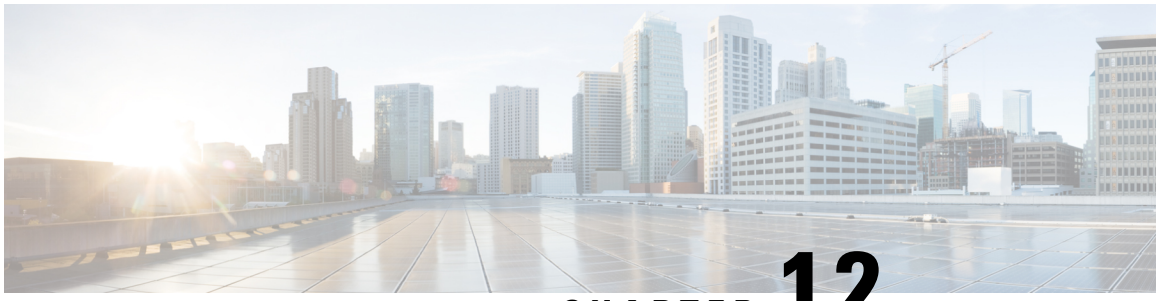
Feature Information for Configuring EVC Quality of Service

The following table provides release information about the feature or features described in this module. This table lists only the software release that introduced support for a given feature in a given software release train. Unless noted otherwise, subsequent releases of that software release train also support that feature.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to www.cisco.com/go/cfn. An account on Cisco.com is not required.

Table 11: Feature Information for EVC Quality of Service

Feature Name	Releases	Feature Information
EVC Quality of Service	Cisco IOS XE Release 3.3 Cisco IOS Release 15.5(2)T	This document contains information about how to enable quality of service (QoS) features (such as traffic classification and traffic policing) for use on an Ethernet virtual circuit (EVC). The EVC Quality of Service feature was introduced on the Cisco ASR 1000 Series Aggregation Services Router. The following commands were introduced or modified: **service-policy**, **show policy-map interface service instance**.



CHAPTER 12

Quality of Service for Etherchannel Interfaces

Quality of Service (QoS) is supported on Ethernet Channel (Etherchannel) interfaces on Cisco ASR 1000 Series Routers. The QoS functionality has evolved over several Cisco IOS XE releases and has different capabilities based on software level, Etherchannel configuration, and configured Modular QoS CLI (MQC) features.

Finding Feature Information

Your software release may not support all the features documented in this module. For the latest caveats and feature information, see Bug Search Tool and the release notes for your platform and software release. To find information about the features documented in this module, and to see a list of the releases in which each feature is supported, see the feature information table.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to https://cfnng.cisco.com/. An account on Cisco.com is not required.

Information About QoS for Etherchannels

Etherchannel with QoS Feature Evolution

An Etherchannel is a port-channel architecture that allows grouping of several physical links to create one logical Ethernet link for the purpose of providing fault tolerance, and high-speed links between switches, routers, and servers. An Etherchannel can be created from between two and eight active Fast, Gigabit, or 10-Gigabit Ethernet ports, with an additional one to eight inactive (failover) ports, which become active as the other active ports fail.

QoS for Etherchannel interfaces has evolved over several Cisco IOS XE releases. It is important to understand what level of support is allowed for your current level of Cisco IOS XE software and underlying Etherchannel

configuration. Various combinations of QoS are supported based on how Etherchannel is configured. There are three different modes in which Etherchannel can be configured:

- Etherchannel VLAN-based load balancing via port-channel subinterface encapsulation CLI
- Etherchannel Active/Standby with LACP (no Etherchannel load balancing)
- Etherchannel with LACP with load balancing

Each of these models has specific restrictions regarding which levels of Cisco IOS XE software include support and the possible QoS configurations with each.

The following summarizes the various Etherchannel and QoS configuration combinations that are supported. Example configurations will be provided later in this document. Unless specifically mentioned together, the combination of service policies in different logical and physical interfaces for a given Etherchannel configuration is not supported.

Etherchannel VLAN-Based Load Balancing via Port-Channel Subinterface Encapsulation CLI

Supported in Cisco IOS XE Release 2.1 or later:

- Egress MQC Queuing Configuration on Port-Channel Subinterface
- Egress MQC Queuing Configuration on Port-Channel Member Link
- QoS Policies Aggregation—Egress MQC Queuing at Subinterface
- Ingress Policing and Marking on Port-Channel Subinterface
- Egress Policing and Marking on Port-Channel Member Link

Supported in Cisco IOS XE Release 2.6 or later:

- QoS Policies Aggregation—MQC Support for Multiple Queue Aggregation at Main Interface - Egress MQC Queuing at Main Interface

Etherchannel Active/Standby with LACP (No Etherchannel Load Balancing)

Supported in Cisco IOS XE 2.4 or later:

- Egress MQC Queuing on Port-Channel Member Link—No Etherchannel Load Balancing

Etherchannel with LACP and Load Balancing

Supported in Cisco IOS XE 2.5 or later:

- Egress MQC Queuing Configuration on Port-Channel Member Link—Etherchannel Load Balancing

Supported in Cisco IOS XE 3.12 or later:

- General MQC QoS support on Port-channel main-interface

  We recommend that as a best practice for QoS, that you use port-channel aggregation—see the "Aggregate EtherChannel Quality of Service" chapter.

Supported in Cisco IOS XE 3.16.3 or later and in Cisco IOS XE Fuji 16.3 or later:

- General MQC QoS support on Port-channel sub-interface

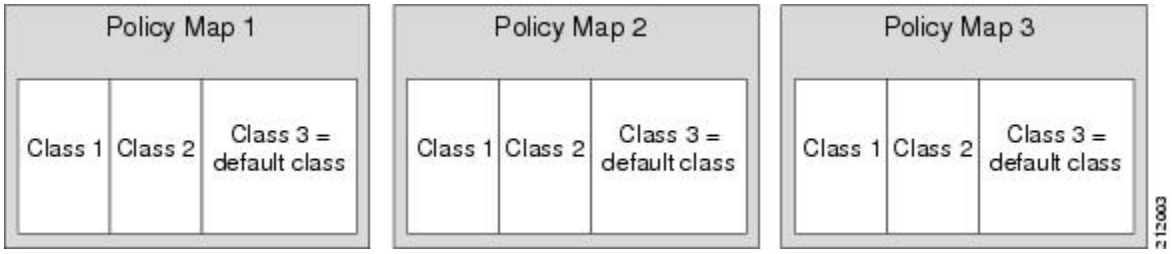
  We recommend that as a best practice for QoS, that you use port-channel aggregation—see the "Aggregate EtherChannel Quality of Service" chapter.

Understanding Fragments in Class Definition Statements

The QoS Policies Aggregation feature introduces the idea of fragments in class definition statements. A default traffic class definition statement can be marked as a fragment within a policy-map. Other policy-maps on the same interface can also define their default traffic class statements as fragments, if desired. A separate policy-map can then be created with a service fragment class definition statement that will be used to apply QoS to all of the fragments as a single group.

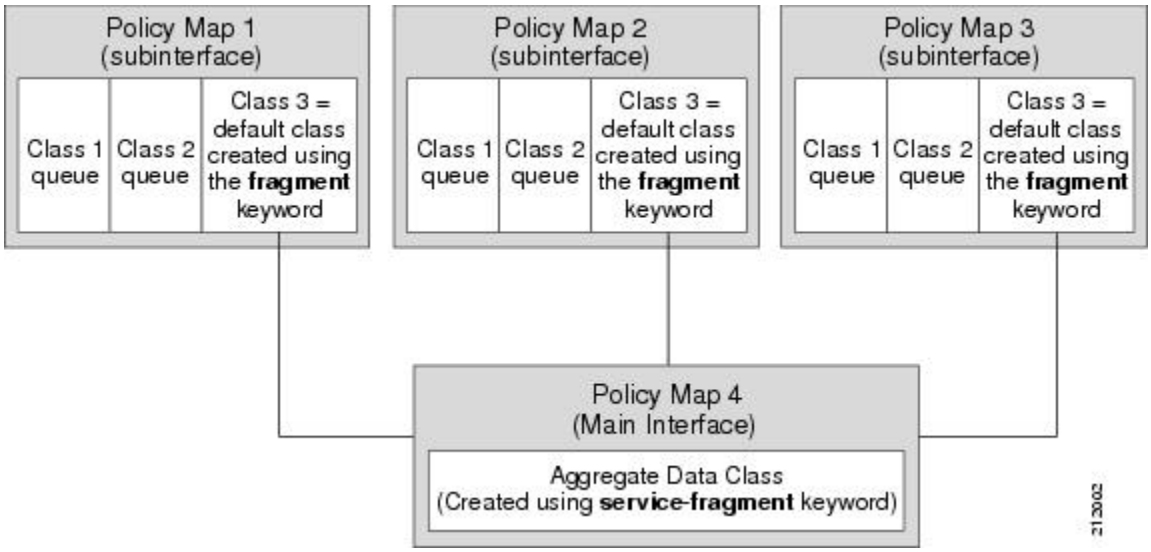
The figure below provides an example of one physical interface with three attached policy-maps that is not using fragments. Note that each policy-map has a default traffic class that can classify traffic only for the default traffic within its own policy-map.

Figure 61: Physical Interface with Policy-Maps—Not Using Fragments



The figure below shows the same configuration configured with fragments, and adds a fourth policy-map with a class definition statement that classifies the fragments collectively. The default traffic classes are now classified as one service fragment group rather than three separate default traffic classes within the individual policy-maps.

Figure 62: Physical Interface with Policy-Maps—Using Fragments



Fragments for Gigabit Etherchannel Bundles

When fragments are configured for Gigabit Etherchannel bundles, the policy-maps that have a default traffic class configured using the **fragment** keyword are attached to the member subinterface links, and the policy-maps

that have a traffic class configured with the **service-fragment** keyword to collectively classify the fragments is attached to the physical interface.

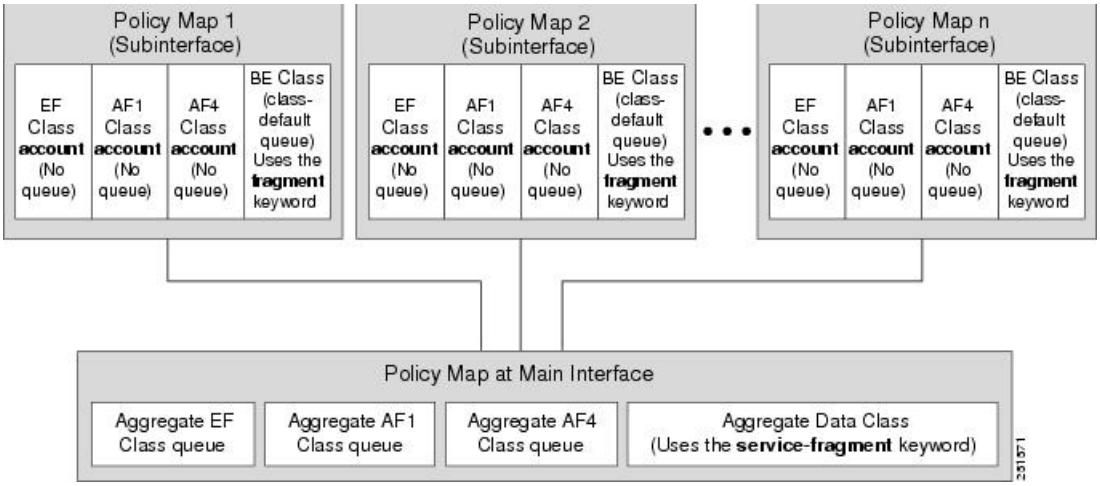
All port-channel subinterfaces configured with fragments that are currently active on a given port-channel member link will use the aggregate service fragment class on that member link. If a member link goes down, the port-channel subinterfaces that must switch to the secondary member link will then use the aggregate service fragment on the new interface.

QoS: Policies Aggregation MQC

The QoS: Policies Aggregation MQC Support for Multiple Queue Aggregation at Main Interface feature extends the previous support of aggregation of class-default traffic using the **fragment** and **service-fragment** configurations, to other user-defined traffic classes in a subinterface policy-map, such as DSCP-based traffic classes, that are aggregated at the main-interface policy-map as shown in the figure below.

When no queueing is configured on a traffic class in the subinterface policy-map, the **account** command can be used to track queueing drops that occur at the aggregate level for these classes, and can be displayed using the **show policy-map interface** command.

Figure 63: Policy-Map Overview for the MQC Support for Multiple Queue Aggregation at Main Interface Feature



Differences Between the Original Feature and the MQC Support for Multiple Queue AggregationDifferences Between Policy Aggregation—Egress MQC Queuing at Subinterface and the MQC Support for Multiple Queue Aggregation at Main Interface

Although some of the configuration between the “Policy Aggregation – Egress MQC Queuing at Subinterface” scenario and the “MQC Support for Multiple Queue Aggregation at Main Interface - Egress MQC Queuing at Main Interface” scenario appear similar, there are some important differences in the queuing behavior and the internal data handling. See the figure in the “Understanding the QoS: Policies Aggregation MQC” section.

For example, both configurations share and require the use of the **fragment** keyword for the **class class-default** command in the subscriber policy-map, as well as configuration of the **service-fragment** keyword for a user-defined class in the main-interface policy-map to achieve common policy treatment for aggregate traffic.

However, the use of this configuration results in different behavior between the original and enhanced QoS policies aggregation implementation:

- In the original implementation using the fragment and service-fragment architecture, all default class traffic and any traffic for classes without defined queueing features at the subinterface goes to the class-default queue and is aggregated into a common user-defined queue and policy defined at the main policy-map. Subinterface traffic aggregation (for example, from multiple subscribers on the same physical interface) ultimately occurs only for a single class, which is the default class.
- In the enhanced implementation of the MQC Support for Multiple Queue Aggregation at Main Interface feature also using the fragment and service-fragment architecture, all default class traffic also goes to the class-default queue and is aggregated into a common user-defined queue and policy defined at the main policy-map. However, other classes, such as DSCP-based subscriber traffic classes, are also supported for an aggregate policy. These traffic classes do not support any queues or queueing features other than **account** at the subscriber policy-map. The use of the fragment and service-fragment architecture enables these other subscriber traffic classes (from multiple subscribers on the same physical interface) to achieve common policy treatment for aggregate traffic that is defined for those same classes at the main policy-map.

How to Configure QoS for Etherchannels

Configuring Egress MQC Queuing on Port-Channel Subinterface

Before you begin

Traffic classes must be configured using the **class-map** command. A one- or two-level hierarchical policy-map should be configured using previously defined class maps. The port-channel subinterface should have been previously configured with the appropriate encapsulation subcommand to match the select primary and secondary physical interfaces on the Etherchannel. Cisco IOS XE Release 2.1 or later software is required. The global configuration must contain the **port-channel load-balancing vlan-manual** command, or the port-channel main-interface configuration must contain the **load-balancing vlan** command. It is assumed that these commands have already been executed.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **interface port-channel** *port-channel-number* **.** *subinterface-number*
4. **service-policy output** *policy-map-name*
5. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.

	Command or Action	Purpose
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**interface port-channel** *port-channel-number* **.** *subinterface-number* **Example:** `Device(config)# interface port-channel 1.200`	Specifies the port-channel subinterface that receives the service policy configuration.

Step 4	**service-policy output** *policy-map-name*
	Example:
	`Device(config-subif)#`
	`service-policy output WAN-GEC-sub-Out`
		Specifies the name of the service policy that is applied to output traffic.

Step 5	**end** **Example:** `Device(config-subif)# end`	Exits interface configuration mode and returns to privileged EXEC mode.

Configuring Egress MQC queuing on Port-Channel Member Links

Before you begin

Traffic classes must be configured using the **class-map** command. A one- or two-level hierarchical policy-map that uses queuing features should be configured using previously defined class maps. The Etherchannel member link interface should already be configured to be part of the channel group (Etherchannel group). No policy-maps that contain queuing commands should be configured on any port-channel subinterfaces. Cisco IOS XE Release 2.1 or later software is required. The global configuration must contain the **port-channel load-balancing vlan-manual** command, or the port-channel main-interface configuration must contain the **load-balancing vlan** command. It is assumed that these commands have already been executed.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **interface GigabitEthernet** *card/bay/port*
4. **service-policy output** *policy-map-name*
5. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable**	Enables privileged EXEC mode.

	Command or Action	Purpose
	Example: `Device> enable`	• Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**interface GigabitEthernet** *card/bay/port* **Example:** `Device(config)# interface GigabitEthernet 0/1/0`	Specifies the member link physical interface that receives the service policy configuration.

Step 4	**service-policy output** *policy-map-name*
	Example:
	`Device(config-if)#`
	`service-policy output WAN-GEC-sub-Out`
		Specifies the name of the service policy that is applied to output traffic for this physical interface that is part of the Etherchannel.

Step 5	**end** **Example:** `Device(config-if)# end`	Exits interface configuration mode and returns to privileged EXEC mode.

Configuring QoS Policies Aggregation—Egress MQC Queuing at Subinterface

Before you begin

Default class traffic from multiple Port-channel subinterfaces can be aggregated into a common policy-map at the main interface when you use the **fragment** keyword at the subinterface **class class-default** configuration, and the **service-fragment** configuration at the main interface class. Queuing occurs at the subinterface for other traffic classes that are defined with queuing features in the subinterface policy-map.

This feature is configured using Modular QoS CLI (MQC). It is most useful in QoS configurations where several policy-maps attached to the same physical interface want aggregated treatment of multiple default traffic classes from multiple port-channel sub-interfaces. Cisco IOS XE Release 2.1 or later software is required. The global configuration must contain the **port-channel load-balancing vlan-manual** command, or the port-channel main-interface must have the **load-balancing vlan** command. It is assumed that these commands have already been executed.

Note This feature is supported when policy-maps are attached to multiple port-channel subinterfaces and the port-channel member link interfaces. This feature cannot be used to collectively classify default traffic classes of policy-maps on different physical interfaces. It can collectively classify all traffic directed toward a given port-channel member link when designated by the **primary** or **secondary** directives on the subinterface **encapsulation** command. All subinterface traffic classes should have queues. However, when a traffic class in the subinterface policy-map is not configured with any queuing feature (commands such as **priority**, **shape**, **bandwidth**, **queue-limit**, **fair-queue**, or **random-detect**), the traffic is assigned to the class-default queue. No classification occurs or is supported at the main interface policy-map for any subinterface traffic classes that do not use the **fragment** and **service-fragment** configuration.

A multistep process is involved with the complete configuration of the QoS Policies Aggregation feature. The following sections detail those steps.

Note the following about attaching and removing a policy-map:

- To configure QoS Policies Aggregation, you must attach the policy-map that contains the **service-fragment** keyword to the main interface first, and then you must attach the policy-map that contains the **fragment** keyword to the subinterface.
- To disable QoS Policies Aggregation, you must remove the policy-map that contains the fragment keyword from the subinterface first, and then you must remove the policy-map that contains the service-fragment keyword from the main interface.

Configuring a Fragment Traffic Class in a Policy-Map

Before you begin

This procedure shows only how to configure the default traffic class as a fragment within a policy-map. It does not include steps on configuring other classes within the policy-map, or other policy-maps on the device.

Example

Note This example shows a sample configuration that is supported in releases prior to Cisco IOS XE Release 2.6.

In the following example, a fragment named BestEffort is created in policy-map subscriber1 and policy-map subscriber 2. In this example, queuing features for other traffic classes are supported at the subinterface policy-map.

```
policy-map subscriber1
 class voice
  set cos 5
  priority level 1
 class video
  set cos 4
  priority level 2
 class class-default fragment BestEffort
  shape average 200000000
  bandwidth remaining ratio 10
```

```
policy-map subscriber 2
 class voice
  set cos 5
  priority level 1
 class video
  set cos 4
  priority level 2
 class class-default fragment BestEffort
  shape average 200000000
  bandwidth remaining ratio 10
```

Note This example shows a sample configuration that is supported in Cisco IOS XE Release 2.6 and later releases.

The following example also shows how to configure a fragment named BestEffort for the default class in a policy-map on a subinterface using the QoS Policies Aggregation MQC Support for Multiple Queue Aggregation at Main Interface implementation. In this example, notice that queuing features are not supported for the other classes in the policy-map:

```
policy-map subscriber1
 class voice
  set cos 5
  account
 class video
  set cos 4
  account
 class AF1
  account
 class class-default fragment BestEffort
  shape average 200000000
  bandwidth remaining ratio 10
```

After configuring default class statements as fragments in multiple subinterface policy-maps, a separate policy-map with a class statement using the service-fragment keyword must be configured to apply QoS to the class stratements configured as fragments.

What to Do Next

After configuring multiple default class statements as fragments in a policy-map, a separate policy-map with a class statement using the **service-fragment** keyword must be configured to apply QoS to the class statements configured as fragments.

This process is documented in the “Configuring a Service Fragment Traffic Class” section.

Configuring a Service Fragment Traffic Class

Before you begin

This task describes how to configure a service fragment traffic class statement within a policy-map. A service fragment traffic class is used to apply QoS to a collection of default class statements that have been configured previously in other policy-maps as fragments.

This procedure assumes that fragment default traffic classes were already created. The procedure for creating fragment default traffic classes is documented in the “Configuring a Fragment Traffic Class in a Policy-Map” section.

Like any policy-map, the configuration does not manage network traffic until it has been attached to an interface. This procedure does not cover the process of attaching a policy-map to an interface.

Note A service fragment can be used to collectively classify fragments only from the same physical interface. Fragments from different interfaces cannot be classified using the same service fragment.

Only queueing features are allowed in classes where the **service-fragment** keyword is entered, and at least one queueing feature must be entered in classes when the **service-fragment** keyword is used.

A policy-map with a class using the **service-fragment** keyword can be applied only to traffic leaving the interface (policy-maps attached to interfaces using the **service-policy output** command).

A class configured using the **service-fragment** keyword cannot be removed when it is being used to collectively apply QoS to fragments that are still configured on the interface. If you wish to remove a class configured using the **service-fragment** keyword, remove the fragment traffic classes before removing the service fragment.

The **service-fragment** keyword cannot be entered in a child policy-map.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **policy-map** *policy-map-name*
4. **class** *class-name* **service-fragment** *fragment-class-name*
5. **shape average percent** *percent*
6. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**policy-map** *policy-map-name* **Example:** `Device(config)# policy-map BestEffortFragments`	Specifies the name of the traffic policy to configure and enters policy-map configuration mode.
Step 4	**class** *class-name* **service-fragment** *fragment-class-name* **Example:**	Specifies a class of traffic that is the composite of all fragments matching the *fragment-class-name*. The *fragment-class-name* when defining the fragments in other policy-maps must match the *fragment-class-name* in this

	Command or Action	Purpose
	`Device(config-pmap)# class data service-fragment BestEffort`	command line to properly configure the service fragment class.
Step 5	**shape average percent** *percent* **Example:** `Device(config-pmap-c)# shape average percent 50`	Enters a QoS configuration command. Only queueing features are supported in default traffic classes configured as fragments. The queueing features that are supported are **bandwidth**, **shape**, and **random-detect exponential-weighting-constant**. Multiple QoS queueing commands can be entered.
Step 6	**end** **Example:** `Device(config-pmap-c)# end`	Exits policy-map class configuration mode and returns to privileged EXEC mode.

Examples

Note This example shows a sample configuration that is supported in releases prior to Cisco IOS XE Release 2.6.

In the following example, a policy-map is created to apply QoS to all fragments named BestEffort.

```
policy-map main-interface
 class data service-fragment BestEffort
  shape average 400000000
```

In the following example, two fragments are created and then classified collectively using a service fragment.

```
policy-map subscriber1
 class voice
  set cos 5
  priority level 1
 class video
  set cos 4
  priority level 2
 class class-default fragment BestEffort
  shape average 200000000
  bandwidth remaining ratio 10
policy-map subscriber 2
 class voice
  set cos 5
  priority level 1
 class video
  set cos 4
  priority level 2
 class class-default fragment BestEffort
```

```
  shape average 200000000
  bandwidth remaining ratio 10
```

Note This example shows a sample configuration that is supported in Cisco IOS XE Release 2.6 and later releases.

The following example shows the creation of two fragments called BestEffort in the subinterface policy-maps, followed by a sample configuration for the **service-fragment** called BestEffort to aggregate the queues at the main interface policy-map:

```
policy-map subscriber1
 class voice
  set cos 5
  account
 class video
  set cos 4
  account
 class AF1
  account
 class class-default fragment BestEffort
  shape average 200000000
  bandwidth remaining ratio 10
policy-map subscriber2
 class voice
  set cos 5
  account
 class video
  set cos 4
  account
 class AF1
  account
 class class-default fragment BestEffort
  shape average 200000000
  bandwidth remaining ratio 10
policy-map main-interface
 class voice
  priority level 1
 class video
  priority level 2
 class AF1
  bandwidth remaining ratio 90
 class data service-fragment BestEffort
  shape average 400000000
  bandwidth remaining ratio 1
```

Troubleshooting Tips

Ensure that all class statements that should be part of the same service fragment share the same *fragment-class-name*.

What to Do Next

Attach the service fragment traffic classes to the main physical interfaces.

Attach the fragment traffic classes to the member-link subinterfaces.

Configuring Service Fragments on a Physical Interface Supporting a Gigabit Etherchannel Bundle

Before you begin

This procedure assumes that a service fragment traffic class has already been created. A service fragment traffic class cannot be configured without configuring a fragment class. The procedure for creating a fragment class is documented in the "Configuring a Fragment Traffic Class in a Policy-Map" section. The procedure for creating a service fragment traffic classes is documented in the "Configuring a Service Fragment Traffic Class" section.

These instructions do not provide any details about the options that can be configured for Gigabit Etherchannel member link subinterfaces. These instructions document only the procedure for attaching a policy-map that already has a fragment traffic class to a member link subinterface.

Note For proper behavior, when a port-channel member link goes down, all member links should have the same policy-map applied.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **interface GigabitEthernet** *card/bay/port*
4. **service-policy output** *service-fragment-class-name*
5. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**interface GigabitEthernet** *card/bay/port* **Example:** `Device(config)# interface GigabitEthernet 0/1/0`	Specifies the member link physical interface that receives the service-policy configuration.

	Command or Action	Purpose
Step 4	**service-policy output** *service-fragment-class-name* **Example:** `Device(config-if)# service-policy output aggregate-member-link`	Attaches a service policy that contains a service fragment default traffic class to the physical Gigabit Ethernet interface.
Step 5	**end** **Example:** `Device(config-if)# end`	Exits interface configuration mode and returns to privileged EXEC mode.

Examples

In the following example, the policy-map aggregate-member-link is attached to the physical interface.

```
interface GigabitEthernet1/1/1
 service-policy output aggregate-member-link
!
interface GigabitEthernet1/1/2
 service-policy output aggregate-member-link
```

What to do next

Ensure that the fragment class name is consistent across service-fragment and fragment class definitions. Continue to the "Configuring Fragments on Gigabit Etherchannel Member Link Subinterfaces" section.

Configuring Fragments on Gigabit Etherchannel Member Link Subinterfaces

Before you begin

This procedure assumes that a service fragment traffic class has already been created. A service fragment traffic class cannot be configured without configuring a fragment class. The procedure for creating a fragment class is documented in the "Configuring a Fragment Traffic Class in a Policy-Map" section. The procedure for creating a service fragment traffic class is documented in the "Configuring a Service Fragment Traffic Class" section.

These instructions do not provide any details about the options that can be configured for Gigabit Etherchannel member link subinterfaces. These instructions only document the procedure for attaching a policy-map that already has a fragment traffic class to a member link subinterface.

Fragments cannot be used for traffic on two or more physical interfaces.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **interface port-channel** *port-channel-interface-number* **.** *port-channel-subinterface-number*
4. **service-policy output** *fragment-class-name*

5. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**interface port-channel** *port-channel-interface-number* **.** *port-channel-subinterface-number* **Example:** `Device(config)# interface port-channel 1.100`	Enters subinterface configuration mode to configure an Etherchannel member link subinterface.
Step 4	**service-policy output** *fragment-class-name* **Example:** `Device(config-subif)# service-policy output subscriber`	Attaches a service policy that contains a fragment default traffic class to the Etherchannel member link subinterface
Step 5	**end** **Example:** `Device(config-subif)# end`	Exits subinterface configuration mode and returns to privileged EXEC mode.

Example

In the following example, the service policy named subscriber has a fragment default traffic class and is attached to the port-channel subinterface of an Etherchannel bundle.

```
interface port-channel 1.100
 service-policy output subscriber
```

Configuring Ingress Policing and Marking on Port-Channel Subinterface

Before you begin

Traffic classes must be configured using the **class-map** command. A one- or two-level hierarchical policy-map should be configured using previously defined class maps. The Etherchannel member link interface should

already be configured to be part of the channel group (Etherchannel group). Cisco IOS XE Release 2.1 or later software is required. The global configuration must contain the **port-channel load-balancing vlan-manual** command or the port-channel main-interface configuration must contain the **load-balancing vlan** command. It is assumed that these commands have already been executed.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **interface port-channel** *port-channel-number* **.** *port-channel-interface-number* **.** *sub-interface-number*
4. **service-policy input** *policy-map-name*
5. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**interface port-channel** *port-channel-number . port-channel-interface-number . sub-interface-number* **Example:** `Device(config)# interface port-channel 1.100.100`	Enters subinterface configuration mode to configure an Etherchannel member link subinterface.
Step 4	**service-policy input** *policy-map-name* **Example:** `Device(config-subif)# service-policy input sub-intf-input`	Specifies the name of the service policy that is applied to input traffic for the port-channel subinterface previously specified.
Step 5	**end** **Example:** `Device(config-subif)# end`	Exits subinterface configuration mode and returns to privileged EXEC mode.

Example

In the following example, the service policy named sub-intf-input is defined and attached to the port-channel subinterface in the input direction.

```
policy-map sub-intf-input
  class voice
    set precedence 5
  class video
    set precedence 6
  class class-default
    set precedence 3
!
interface Port-channel 1.100
 service-policy input sub-intf-input
```

Configuring Egress Policing and Marking on Port-Channel Member Links

Before you begin

Traffic classes must be configured using the **class-map** command. A one- or two-level hierarchical policy-map should be configured using previously defined class maps. The Etherchannel member link interface should already be configured to be part of the channel group (Etherchannel group). Cisco IOS XE Release 2.1 or later software is required. The global configuration must contain the **port-channel load-balancing vlan-manual** command or the port-channel main-interface configuration must contain the **load-balancing vlan** command. It is assumed that these commands have already been executed.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **interface port-channel** *port-channel-number . port-channel-interface-number . sub-interface-number*
4. **service-policy output** *policy-map-name*
5. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**interface port-channel** *port-channel-number . port-channel-interface-number . sub-interface-number* **Example:** `Device(config)# interface port-channel 1.100.100`	Enters subinterface configuration mode to configure an Etherchannel member link subinterface.

	Command or Action	Purpose
Step 4	**service-policy output** *policy-map-name* **Example:** `Device(config-subif)# service-policy output WAN-GEC-member-Out-police`	Specifies the name of the service policy that is applied to output traffic for the Etherchannel member link subinterface specified in the previous step.
Step 5	**end** **Example:** `Device(config-subif)# end`	Exits subinterface configuration mode and returns to privileged EXEC mode.

Example

In the following example, the service policy named WAN-GEC-member-Out-police is defined and attached to the port-channel subinterface in the output direction.

```
policy-map WAN-GEC-member-Out-police
  class voice
    set precedence 5
  class video
    set precedence 6
  class class-default
    set precedence 3
!
interface port-channel 1.100
 service-policy output WAN-GEC-member-Out-police
```

Configuring Policies Aggregation—MQC Support for Multiple Queue Aggregation at Main Interface

Before you begin

This feature is configured using the MQC. It is most useful in QoS configurations where several policy-maps attached to the same physical interface want aggregated treatment of multiple user-defined traffic classes from multiple port-channel subinterfaces. Cisco IOS XE Release 2.6 or later software is required. The global configuration must contain the following command: **port-channel load-balancing vlan-manual** or the main interface of the port-channel being configured must have the following command: **port-channel load-balancing vlan**. It is assumed that these commands have already been executed.

This feature is supported when policy-maps are attached to multiple port-channel subinterfaces and the port-channel member link interfaces. This feature cannot be used to collectively classify default traffic classes of policy-maps on different physical interfaces. It can collectively classify all traffic directed towards a given Port-channel member-link when designated by the **primary** or **secondary** directives on the sub-interface **encapsulation** command. The following items describe the behavior and restrictions on configuring this type of QoS Policy Aggregation with Etherchannel:

- Subinterface traffic classes without configured queuing features do not have queues at the subscriber level

- Default class traffic from multiple subinterfaces can be aggregated into a common policy-map at the main interface when you use the **fragment** keyword at the subinterface **class class-default** configuration, and **service-fragment** configuration at the main interface class
- This configuration additionally enables support for other subinterface traffic classes (such as DSCP-based classes) to be aggregated into a common policy-map at the main interface.
- This feature is enabled by using the **fragment** keyword in the subinterface **class-default** class, and **service-fragment** configuration in the main interface class (this also enables aggregation of the default class.
- Queuing features are not configured at the subinterface policy-map for the other traffic classes.
- Queuing occurs at the main interface policy-map for other subinterface traffic classes as an aggregate.
- Optional tracking of statistics is supported using the **account** command for other traffic classes in the subinterface policy-map.

A multistep process is involved with the complete configuration of QoS multiple queue aggregation at a main interface feature, as follows:

1. Configure default class statements as fragments in multiple subinterface policy-maps as described in the "Configuring a Fragment Traffic Class in a Policy-Map" section.
2. Configure a separate policy-map with a class statement using the **service-fragment** keyword in order to apply QoS to the class statements configured as fragments as described in the "Configuring a Service Fragment Traffic Class" section.
3. Configure service fragment traffic classes and attach them to the main physical interfaces as described in the "Configuring Service Fragments on a Physical Interface Supporting a Gigabit Etherchannel Bundle" section.
4. Configure fragment traffic classes and attach them to the member link subinterfaces as described in the "Configuring Fragments on Gigabit Etherchannel Member Link Subinterfaces" section.

Configuring MQC Queuing on Port-Channel Member Link—No Etherchannel Load Balancing

Before you begin

Traffic classes must be configured using the **class-map** command. A one or two level hierarchical policy-map should be configured using previously defined class maps.

Cisco IOS XE Release 2.4 or later software is required.

The port-channel main interface should also contain the following commands that create an active/standby scenario. Such a configuration will allow only a single interface to be active and forwarding traffic at any time.

- **interface Port-channel1**
- **lacp fast-switchover**
- **lacp max-bundle 1**

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **interface GigabitEthernet** *card/bay/port*
4. **service-policy output** *policy-map-name*
5. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**interface GigabitEthernet** *card/bay/port* **Example:** `Device(config)# interface GigabitEthernet 0/1/0`	Specifies the member link physical interface that receives the service policy configuration.

Step 4	**service-policy output** *policy-map-name*
	Example:
	`Device(config-if)#`
	`service-policy output WAN-GEC-member-Out`
		Specifies the name of the service policy that is applied to output traffic.

Step 5	**end** **Example:** `Device(config-if)# end`	Exits interface configuration mode and returns to privileged EXEC mode.

Example

In the following example, the service policy named main-intf is defined and attached to the port-channel member links in the output direction.

```
interface Port-channel 1
 lcap fast-switchover
 lacp max-bundle 1
!
policy-map main-intf
 class voice
  priority
  police cir 10000000
```

```
 class video
  bandwidth remaining ratio 10
 class class-default
  bandwidth remaining ratio 3
!
interface GigabitEthernet0/0/0
 channel-group 1 mode active
 service-policy output main-intf
!
interface GigabitEthernet0/0/1
 channel-group 1 mode active
 service-policy output main-intf
```

Configuring MQC Queuing Configuration on Port-Channel Member Link—Etherchannel Load Balancing

Before you begin

Traffic classes must be configured using the **class-map** command. A one- or two-level hierarchical policy-map should be configured using previously defined class maps. The port-channel subinterface should have been previously configured with the appropriate encapsulation subcommand to match the select primary and secondary physical interfaces on the Etherchannel. Cisco IOS XE Release 2.5 or later software is required.

The Etherchannel setup may have multiple active interfaces with flow-based load balancing enabled.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **interface GigabitEthernet** *card/bay/port*
4. **service-policy output** *policy-map-name*
5. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**interface GigabitEthernet** *card/bay/port* **Example:** `Device(config)# interface GigabitEthernet 0/1/0`	Specifies the member link physical interface that receives the service policy configuration.

	Command or Action	Purpose
Step 4	**service-policy output** *policy-map-name* **Example:** `Device(config-if)# service-policy output WAN-GEC-member-Out`	Specifies the name of the service policy that is applied to output traffic.
Step 5	**end** **Example:** `Device(config-if)# end`	Exits interface configuration mode and returns to privileged EXEC mode.

Example

In the following example, the service policy named main-intf is defined and attached to the port-channel member links in the output direction.

```
  class voice
   priority
   police cir 10000000
  class video
   bandwidth remaining ratio 10
  class class-default
   bandwidth remaining ratio 3
!
interface GigabitEthernet0/0/0
 channel-group 1 mode active
 service-policy output main-intf
!
interface GigabitEthernet0/0/1
 channel-group 1 mode active
 service-policy output main-intf
```

Configuration Examples for QoS for Etherchannels

Example: Configuring QoS Policies Aggregation—Egress MQC Queuing at Subinterface

```
port-channel load-balancing vlan-manual
!
class-map match-all BestEffort
!
class-map match-all video
 match precedence 4
!
class-map match-all voice
 match precedence 5
!
```

```
policy-map subscriber
 class voice
  priority level 1
 class video
  priority level 2
 class class-default fragment BE
  shape average 100000000
  bandwidth remaining ratios 80

policy-map aggregate-member-link
 class BestEffort service-fragment BE
 shape average 100000000
!
interface Port-channel1
 ip address 209.165.200.225 255.255.0.0
!
interface Port-channel1.100
 encapsulation dot1Q 100
 ip address 209.165.200.226 255.255.255.0
 service-policy output subscriber
!
interface Port-channel1.200
 encapsulation dot1Q 200
 ip address 209.165.200.227 255.255.255.0
 service-policy output subscriber
!
interface Port-channel1.300
 encapsulation dot1Q 300
 ip address 209.165.200.228 255.255.255.0
 service-policy output subscriber
!
interface GigabitEthernet1/1/1
 no ip address
 channel-group 1 mode on
 service-policy output aggregate-member-link
!
interface GigabitEthernet1/1/2
 no ip address
 channel-group 1 mode on
 service-policy output aggregate-member-link
```

Example: Configuring QoS Policies Aggregation—MQC Support for Multiple Queue Aggregation at Main Interface

```
port-channel load-balancing vlan-manual
!
policy-map subscriber1
 class voice
  set cos 5
  account
 class video
  set cos 4
  account
 class AF1
  account
 class class-default fragment BestEffort
  shape average 200000000
  bandwidth remaining ratio 10
!
policy-map subscriber2
```

```
 class voice
  set cos 2
  account
 class video
  set cos 3
  account
 class AF1
  account
 class class-default fragment BestEffort
  shape average 200000000
  bandwidth remaining ratio 10
!
policy-map main-interface-out
 class voice
  priority level 1
 class video
  priority level 2
 class AF1
  bandwidth remaining ratio 90
 class data service-fragment BestEffort
  shape average 400000000
  bandwidth remaining ratio 1
!
interface GigabitEthernet1/1/1
 no ip address
 channel-group 1 mode on
 service-policy output main-interface-out
!
interface GigabitEthernet1/1/2
 no ip address
 channel-group 1 mode on
 service-policy output main-interface-out
!
interface Port-channel1.100
 encapsulation dot1Q 100
 ip address 10.0.0.1 255.255.255.0
 service-policy output subscriber1
!
interface Port-channel1.200
 encapsulation dot1Q 200
 ip address 10.0.0.2 255.255.255.0
 service-policy output subscriber2
!
interface Port-channel1.300
 encapsulation dot1Q 300
 ip address 10.0.0.4 255.255.255.0
 service-policy output subscriber2
```

Additional References

Related Documents

Related Topic	Document Title
Cisco IOS commands	*Cisco IOS Master Command List, All Releases*

Related Topic	Document Title
QoS commands: complete command syntax, command modes, command history, defaults, usage guidelines, and examples	*Cisco IOS Quality of Service Solutions Command Reference*
Modular Quality of Service Command-Line Interface	"Applying QoS Features Using the MQC" module
Configuring RADIUS-based policing	*Intelligent Services Gateway Configuration Guide*
CISCO ASR 1000 Series software configuration	*Cisco ASR 1000 Series Aggregation Services Routers Software Configuration Guide*

Technical Assistance

Description	Link
The Cisco Support and Documentation website provides online resources to download documentation, software, and tools. Use these resources to install and configure the software and to troubleshoot and resolve technical issues with Cisco products and technologies. Access to most tools on the Cisco Support and Documentation website requires a Cisco.com user ID and password.	http://www.cisco.com/cisco/web/support/index.html

Feature Information for Quality of Service for Etherchannel Interfaces

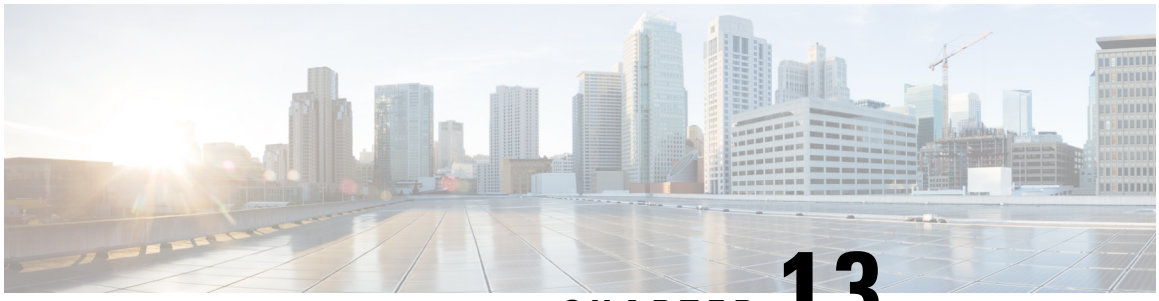
The following table provides release information about the feature or features described in this module. This table lists only the software release that introduced support for a given feature in a given software release train. Unless noted otherwise, subsequent releases of that software release train also support that feature.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to www.cisco.com/go/cfn. An account on Cisco.com is not required.

Table 12: Feature Information for Quality of Service for Etherchannel Interfaces

Feature Name	Releases	Feature Information
Egress MQC Queuing Configuration on Port-Channel Subinterface	Cisco IOS XE Release 2.1	This feature supports the configuration of Egress MQC queuing on port-channel subinterface. This feature was introduced on Cisco ASR 1000 Series Routers.

Feature Name	Releases	Feature Information
Egress MQC Queuing Configuration on Port-Channel Member Link	Cisco IOS XE Release 2.1	This feature supports the configuration of Egress MQC queuing on port-channel member link. This feature was introduced on Cisco ASR 1000 Series Routers.
QoS Policies Aggregation—Egress MQC Queuing at Subinterface	Cisco IOS XE Release 2.1	This feature supports the configuration of QoS Policies Aggregation - Egress MQC queuing at subinterface. This feature was introduced on Cisco ASR 1000 Series Routers.
Ingress Policing and Marking on Port-Channel Subinterface	Cisco IOS XE Release 2.1	This feature supports the configuration of Ingress Policing and Marking on port-channel subinterface. This feature was introduced on Cisco ASR 1000 Series Routers.
Egress Policing and Marking on Port-Channel Member Link	Cisco IOS XE Release 2.1	This feature supports the configuration of Egress policing and marking on port-channel member link. This feature was introduced on Cisco ASR 1000 Series Routers.
Egress MQC Queuing Configuration on Port-Channel Member Link - No Etherchannel Load Balancing	Cisco IOS XE Release 2.4	This feature supports the configuration of Egress MQC Queuing on Port-Channel Member Link - no Etherchannel Load Balancing. This feature was introduced on Cisco ASR 1000 Series Routers.
Egress MQC Queuing Configuration Supported on Port-Channel Member Link - Etherchannel Load Balancing	Cisco IOS XE Release 2.5	This feature supports the configuration of Egress MQC Queuing on Port-Channel Member Link - Etherchannel Load Balancing. This feature was introduced on Cisco ASR 1000 Series Routers.
QoS Policies Aggregation - MQC Support for Multiple Queue Aggregation at Main Interface - Egress MQC Queuing at Main Interface	Cisco IOS XE Release 2.6	This feature supports the configuration of QoS Policies Aggregation - MQC Support for Multiple Queue Aggregation at Main Interface - Egress MQC Queuing at Main Interface. This feature was introduced on Cisco ASR 1000 Series Routers.



CHAPTER 13

Aggregate EtherChannel Quality of Service

The Aggregate EtherChannel Quality of Service (QoS) feature allows you to apply an aggregate egress-queuing policy-map on a port-channel main interface or subinterface. This feature enables QoS support on the aggregate port-channel main interface for the Cisco ASR 1000 Series Aggregation Services Routers.

Restrictions for Aggregate EtherChannel Quality of Service

- The configuration of QoS on Ethernet Virtual Circuit (EVC) with an aggregate port-channel interface is not supported.
- Point-to-Point Protocol over Ethernet (PPPoE) and IP over Ethernet (IPoE) sessions in the context of the Intelligent Services Gateway (ISG) and Intelligent Wireless Access Gateway (iWAG) (with or without QoS) across an aggregate port-channel interface is not supported.
- Virtual Private LAN Services (VPLS) with QoS on an aggregate port-channel interface is not supported.
- Xconnect with QoS on an aggregate port-channel interface is not supported.
- The use of fragment and service-fragment Modular QoS CLI (MQC) keywords in conjunction with the aggregate port-channel interface type is not supported.
- The aggregate-type port-channel interfaces have the following limitations:
  - All the member links of a port channel must be of the same speed. This prevents a potential packet reordering issue. It is not supported to combine Gigabit Ethernet, Fast Ethernet, or Ethernet interfaces into the same port channel.

- 10-Gigabit Ethernet is supported in Cisco IOS XE 3.16.3 or later (it is not supported in Cisco IOS XE 3.17). 10-Gigabit Ethernet is also supported in Cisco IOS XE Denali 16.3 and later.

- MPOL policy applied on both aggregate port-channel main interface and port-channel sub-interface is not supported by any Cisco IOS XE 3S release and is not supported on Cisco IOS XE Everest 16.5.x or earlier.
- QoS on an aggregate port-channel subinterface is not supported for Cisco IOS XE 3.16.2 or earlier (and it is also not supported in Cisco IOS XE 3.17).

Restrictions for PPPOE Session QoS over Aggregate EtherChannel

- All the member links of a port channel must be of the same speed. This prevents a potential packet reordering issue. It is not supported to combine Gigabit Ethernet, Fast Ethernet, or Ethernet interfaces into the same port channel.
- MPOL policy that is applied on both aggregate port-channel main interface and port-channel sub-interface is not supported.
- MPOL policy applied on both aggregate port-channel interface and PPPOE session is supported. The main interface or sub-interface QoS service policy is limited to only a class-default shaper (it can only contain the class class-default and shape command). Additional QoS configurations are not supported on the main interface or sub-interface when QoS service policies are applied to the main or sub-interface and the PPPOE session simultaneously.
- Before PPPOE session QoS is applied, the following command is required:

  platform qos port-channel-aggregate *port-channel interface*

  If the port-channel is already configured in any form, the above command fails.
- The QoS policy can be applied to an aggregate port-channel interface subject to the following scalability limits:
  - Upto 8 port channels
  - Upto 4 member links in a port channel
  - Member links can be split across multiple shared port adapters (SPAs) and SPA interface processor (SIP) cards

Information About Aggregate EtherChannel Quality of Service

Supported Features for Aggregate EtherChannel Quality of Service

The Aggregate EtherChannel Quality of Service feature supports:

- Flow-based load balancing
- Up to three levels of hierarchy
- Configuration of shaping, absolute bandwidth, and relative bandwidth

- A minimum amount of bandwidth for subclasses (VLANs)
- Input QoS (policing and marking) and output QoS (all queuing features) that are enabled simultaneously on an aggregate port-channel main interface and subinterface

Unsupported Feature Combinations for Aggregate EtherChannel Quality of Service

The following combinations of tunnel-type interfaces with QoS are not supported:

- Generic Routing Encapsulation (GRE) tunnels with queuing policy-maps applied, which egress via a port channel with aggregate queuing
- Static virtual tunnel interface (SVTI) and dynamic virtual tunnel interface (DVTI) with queuing QoS applied, which egress via a port channel with aggregate queuing
- Sub-interface belongs to service group and sub-interface applied with service-policy cannot be configured on the same aggregate port-channel simultaneously
- MPOL - policy applied on both aggregate port-channel main interface and port-channel sub-interface

Note Tunnels without queuing QoS (described above) are supported, but are not recommended because hashing algorithms may overload a given physical interface without adequate diversity in IP addresses.

Scalability for Aggregate EtherChannel Quality of Service

The QoS policy can be applied to an aggregate port-channel interface subject to the following scalability limits:

- Up to 8 port channels
- Up to 4 member links in a port channel
- Member links can be split across multiple shared port adapters (SPAs) and SPA interface processor (SIP) cards

How to Configure Aggregate EtherChannel Quality of Service

This procedure describes how to configure Aggregate EtherChannel QoS on the Cisco ASR 1000 Series Aggregation Services Routers.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **platform qos port-channel-aggregate** *port-channel-number*
4. **interface port-channel** *port-channel-number*

5. **service-policy** {**output**} *policy-map*
6. **service-policy** {**input**} *policy-map*

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Router> enable`	Enables privileged EXEC mode. Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Router# configure terminal`	Enters global configuration mode.
Step 3	**platform qos port-channel-aggregate** *port-channel-number* **Example:** `router(config)# platform qos port-channel-aggregate 1`	Enables the aggregate port-channel interface.
Step 4	**interface port-channel** *port-channel-number* **Example:** `router(config)# interface port-channel 1`	Enters interface configuration mode to configure a specific port channel.
Step 5	**service-policy** {**output**} *policy-map* **Example:** `router(config-if)# service-policy output egress_policy`	Attaches a policy-map to an output interface to be used as the service policy for that interface.
Step 6	**service-policy** {**input**} *policy-map* **Example:** `router(config-if)# service-policy input ingress_policy`	Attaches a policy-map to an input interface to be used as the service policy for that interface.

How to Unconfigure Aggregate EtherChannel Quality of Service

This procedure describes how to unconfigure Aggregate EtherChannel QoS on the Cisco ASR 1000 Series Aggregation Services Routers.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **no interface port-channel** *port-channel-number*
4. **no platform qos port-channel-aggregate** *port-channel-number*

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Router> enable`	Enables privileged EXEC mode. Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Router# configure terminal`	Enters global configuration mode.
Step 3	**no interface port-channel** *port-channel-number* **Example:** `router(config)# no interface port-channel 1`	Unconfigures a specific port channel.
Step 4	**no platform qos port-channel-aggregate** *port-channel-number* **Example:** `router(config)# no platform qos port-channel-aggregate 1`	Disables the aggregate port-channel interface and removes the required QoS policies on it.

Configuration Examples for Aggregate EtherChannel Quality of Service

Example: Configuring Aggregate Port-Channel Interface

```
Router# configure terminal
Router(config)# platform qos port-channel-aggregate 1
Router(config)# interface port-channel 1
Router(config-if)# interface GigabitEthernet1/0/1
Router(config-if)# channel-group 1
Router(config-if)# interface GigabitEthernet1/0/0
Router(config-if)# channel-group 1
Router(config-if)# interface port-channel 1.1
Router(config-subif)# encap
Router(config-subif)# encapsulation dot
Router(config-subif)# encapsulation dot1Q 2
Router(config-subif)# ip addr 14.0.1.2 255.255.255.0
Router(config-subif)# interface port-channel 1.2
Router(config-subif)# encapsulation dot1Q 3
Router(config-subif)# ip addr 14.0.2.2 255.255.255.0
Router(config-subif)# interface port-channel 1.3
Router(config-subif)# encapsulation dot1Q 4
Router(config-subif)# ip addr 14.0.3.2 255.255.255.0
Router(config-subif)# end
```

Example: Configuring a Class Map for QoS

```
Router# configure terminal
Router(config)# class-map vlan_2
Router(config-cmap)# match vlan  2
Router(config-cmap)# class-map vlan_3
Router(config-cmap)# match vlan  3
Router(config-cmap)# class-map vlan_4
Router(config-cmap)# match vlan  4
Router(config-cmap)# class-map prec1
Router(config-cmap)# match precedence 1
Router(config-cmap)# class-map prec2
Router(config-cmap)# match precedence 2
Router(config-cmap)# class-map prec3
Router(config-cmap)# match precedence 3
Router(config-cmap)# class-map prec4
Router(config-cmap)# match precedence 4
Router(config-cmap)# end
```

Example: Configuring a Policy-Map for QoS

```
Router# configure terminal
Router(config)# policy-map child-vlan
Router(config-pmap)# class prec1
Router(config-pmap-c)# police cir percent 20
Router(config-pmap-c-police)# exit
Router(config-pmap-c)# priority level 1
Router(config-pmap-c)# class prec2
Router(config-pmap-c)# police cir percent 40
Router(config-pmap-c-police)# exit
Router(config-pmap-c)# priority level 2
Router(config-pmap-c)# class prec3
Router(config-pmap-c)# bandwidth remaining ratio 3
Router(config-pmap-c)# class class-default
Router(config-pmap-c)# bandwidth remaining ratio 1
Router(config-pmap-c)# random-detect
Router(config-pmap-c)#!
Router(config-pmap-c)# policy-map egress_policy
Router(config-pmap)# class vlan_2
Router(config-pmap-c)# shape average 100000000
Router(config-pmap-c)# service-policy child-vlan
Router(config-pmap-c)# class vlan_3
Router(config-pmap-c)# shape average 200000000
Router(config-pmap-c)# service-policy child-vlan
Router(config-pmap-c)# class vlan_4
Router(config-pmap-c)# shape average 300000000
Router(config-pmap-c)# service-policy child-vlan
Router(config-pmap-c)#!
Router(config-pmap-c)# policy-map ingress_policy
Router(config-pmap)# class vlan_2
Router(config-pmap-c)# police cir 80000000
Router(config-pmap-c-police)# conform-action set-prec-transmit 1
Router(config-pmap-c-police)# class vlan_2
Router(config-pmap-c)# set dscp AF21
Router(config-pmap-c)# class class-default
Router(config-pmap-c)# set dscp 0
Router(config-pmap-c)# end
```

Example: Applying QoS to Port Channel Interface

```
Router# configure terminal
Router(config)# interface port-channel 1
Router(config-if)# service-policy output egress_policy
Router(config-if)# service-policy input ingress_policy
Router(config-if)# end
```

How to Configure Aggregate EtherChannel Subinterface Quality of Service

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **platform qos port-channel-aggregate** *port-channel-number*
4. **interface port-channel** *port-channel-number*
5. **interface port-channel** *port-channel-number.subinterface-number*
6. **service-policy** {**output**} *policy-map*
7. **service-policy** {**input**} *policy-map*
8. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**platform qos port-channel-aggregate** *port-channel-number* **Example:** `Device(config)# platform qos port-channel-aggregate 1`	Enables the aggregate port-channel interface.
Step 4	**interface port-channel** *port-channel-number* **Example:** `Device(config)# interface port-channel 1`	Enters interface configuration mode to configure a specific port channel.
Step 5	**interface port-channel** *port-channel-number.subinterface-number*	Enters interface configuration mode to configure a specific port channel subinterface.

	Command or Action	Purpose
	Example: `Device(config)# interface port-channel 1.2`	
Step 6	**service-policy** {**output**} *policy-map* **Example:** `Device(config-if)# service-policy output egress_policy`	Attaches a policy-map to an output interface to be used as the service policy for that interface.
Step 7	**service-policy** {**input**} *policy-map* **Example:** `Device(config-if)# service-policy input ingress_policy`	Attaches a policy-map to an input interface to be used as the service policy for that interface.
Step 8	**end** **Example:** `Device(config)# end`	Exits global configuration mode.

How to Unconfigure Aggregate EtherChannel Subinterface Quality of Service

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **no interface port-channel** *port-channel-number.subinterface*
4. **no platform qos port-channel-aggregate** *port-channel-number*
5. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**no interface port-channel** *port-channel-number.subinterface* **Example:**	Unconfigures a specific port channel subinterface.

	Command or Action	Purpose
	`Device(config)# no interface port-channel 1.2`	
Step 4	**no platform qos port-channel-aggregate** *port-channel-number* **Example:** `Device(config)# no platform qos port-channel-aggregate 1`	Disables the aggregate port-channel interface and removes the required QoS policies on it.
Step 5	**end** **Example:** `Device(config)# end`	Exits global configuration mode.

Configuration Examples for Aggregate EtherChannel Subinterface Quality of Service

Example: Configuring Aggregate Port-Channel Interface and Subinterface

```
Device# configure terminal
Device(config)# platform qos port-channel-aggregate 2
Device(config)# interface port-channel 2
Device(config-if)# interface GigabitEthernet1/1/1
Device(config-if)# channel-group 2
Device(config-if)# interface GigabitEthernet1/1/0
Device(config-if)# channel-group 2
Device(config-if)# interface port-channel 2.200
Device(config-subif)# encapsulation dot1Q 200
Device(config-subif)# ip addr 15.0.1.2 255.255.255.0
Device(config-subif)# interface port-channel 2.300
Device(config-subif)# encapsulation dot1Q 300
Device(config-subif)# ip addr 15.0.2.2 255.255.255.0
Device(config-subif)# end
```

Example: Configuring a Class Map for QoS

```
Device# configure terminal
Device(config)# class-map vlan_2
Device(config-cmap)# match vlan  2
Device(config-cmap)# class-map vlan_3
Device(config-cmap)# match vlan  3
Device(config-cmap)# class-map vlan_4
Device(config-cmap)# match vlan  4
Device(config-cmap)# class-map prec1
Device(config-cmap)# match precedence 1
Device(config-cmap)# class-map prec2
Device(config-cmap)# match precedence 2
Device(config-cmap)# class-map prec3
Device(config-cmap)# match precedence 3
Device(config-cmap)# class-map prec4
```

```
Device(config-cmap)# match precedence 4
Device(config-cmap)# end
```

Example: Configuring a Policy-Map for QoS

```
Device# configure terminal
Device(config)# policy-map subinterface_child
Device(config-pmap)# class prec1
Device(config-pmap-c)# police cir percent 30
Device(config-pmap-c-police)# exit
Device(config-pmap-c)# priority level 1
Device(config-pmap-c)# class prec2
Device(config-pmap-c)# police cir percent 30
Device(config-pmap-c-police)# exit
Device(config-pmap-c)# priority level 2
Device(config-pmap-c)# class prec3
Device(config-pmap-c)# bandwidth remaining ratio 3
Device(config-pmap-c)# class class-default
Device(config-pmap-c)# bandwidth remaining ratio 1
Device(config-pmap-c)#!
Device(config-pmap-c)# policy-map sub_egress_policy
Device(config-pmap-c)# class class-default
Device(config-pmap-c)# shape average 300000000
Device(config-pmap-c)# service-policy subinterface_child
Device(config-pmap-c)#!
Device(config-pmap-c)# policy-map sub_ingress_policy
Device(config-pmap)# class class-default
Device(config-pmap-c)# police cir 80000000
Device(config-pmap-c)# end
```

Example: Applying QoS to Port Channel Subinterface

```
Device# configure terminal
Device(config)# interface port-channel 2.200
Device(config-if)# service-policy output egress_policy
Device(config-if)# service-policy input ingress_policy
Device(config)# interface port-channel 2.300
Device(config-if)# service-policy output egress_policy
Device(config-if)# service-policy input ingress_policy
Device(config-if)# end
```

Additional References

Related Documents

Related Topic	Document Title
Cisco IOS commands	*Cisco IOS Master Commands List, All Releases*
QoS commands	Cisco IOS Quality of Service Solutions Command Reference

MIBs

MIB	MIBs Link
No new or modified MIBs are supported by this feature.	To locate and download MIBs for selected platforms, Cisco software releases, and feature sets, use Cisco MIB Locator found at the following URL: http://www.cisco.com/go/mibs

Technical Assistance

Description	Link
The Cisco Support website provides extensive online resources, including documentation and tools for troubleshooting and resolving technical issues with Cisco products and technologies. To receive security and technical information about your products, you can subscribe to various services, such as the Product Alert Tool (accessed from Field Notices), the Cisco Technical Services Newsletter, and Really Simple Syndication (RSS) Feeds. Access to most tools on the Cisco Support website requires a Cisco.com user ID and password.	http://www.cisco.com/cisco/web/support/index.html

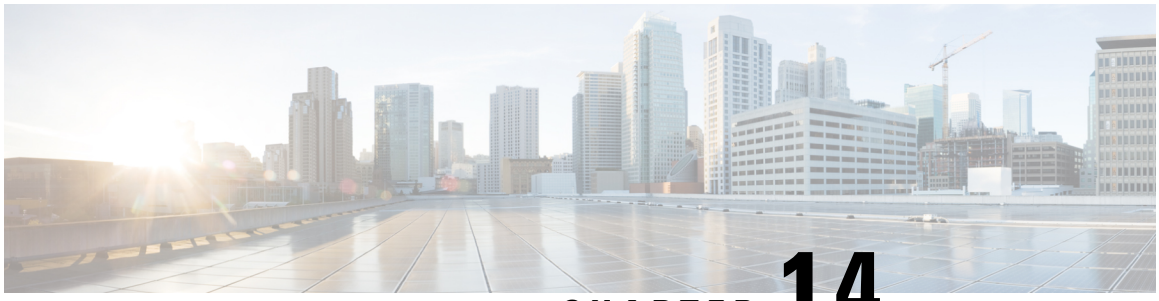
Feature Information for Aggregate EtherChannel Quality of Service

The following table provides release information about the feature or features described in this module. This table lists only the software release that introduced support for a given feature in a given software release train. Unless noted otherwise, subsequent releases of that software release train also support that feature.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to www.cisco.com/go/cfn. An account on Cisco.com is not required.

Table 13: Feature Information for Aggregate EtherChannel Quality of Service

Feature Name	Releases	Feature Information
Aggregate EtherChannel Quality of Service	Cisco IOS XE Release 3.12S	The Aggregate EtherChannel Quality of Service (QoS) feature allows you to apply an aggregate egress-queuing policy-map on a port-channel main interface or subinterface. This feature enables QoS support on the aggregate port-channel main interface for the Cisco ASR 1000 Series Aggregation Services Routers. In Cisco IOS XE Release 3.12S, this feature was implemented on the Cisco ASR 1000 Series Aggregation Services Routers.
Aggregate GEC QoS 10G support	Cisco IOS XE Release 3.16.3S Cisco IOS XE Denali 16.3.1	In Cisco IOS XE Release 3.16.3S, this feature was implemented on the Cisco ASR 1000 Series Aggregation Services Routers.
QoS on GEC portchannel subinterface on ASR1K	Cisco IOS XE Release 3.16.3S Cisco IOS XE Denali 16.3.1	In Cisco IOS XE Release 3.16.3S, this feature was implemented on the Cisco ASR 1000 Series Aggregation Services Routers.
QoS on GEC portchannel subinterface on ISR 4000	Cisco IOS XE Everest 16.6.1	In Cisco IOS XE Everest 16.6.1 release, this feature was implemented on the Cisco ISR 4000 Series Integrated Services Routers.



CHAPTER 14

PPPoGEC Per Session QoS

The PPPoGEC Per Session QoS feature supports the configuration of specific QoS policies on PPPoE sessions on the PPP Termination and Aggregation (PTA), L2TP Access Concentrator (LAC), or L2TP Network Server (LNS) devices in a PPPoE /L2TP environment (broadband deployments). PPPoE sessions with Etherchannel Active/Standby functionality is also supported on Cisco ASR 1000 Series Routers acting as PTA, LAC, or LNS devices in a PPPoE/L2TP environment.

Finding Feature Information

Your software release may not support all the features documented in this module. For the latest caveats and feature information, see Bug Search Tool and the release notes for your platform and software release. To find information about the features documented in this module, and to see a list of the releases in which each feature is supported, see the feature information table.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to https://cfnng.cisco.com/. An account on Cisco.com is not required.

Information About PPPoGEC Per Session QoS

Restrictions for PPPoGEC Per Session QoS

- QoS policy-maps cannot be configured on member links, a port-channel main interface, or a port-channel subinterface that is associated with the transmit path for PPPoE sessions with QoS.

PPPoGEC Sessions with Active/Standby Etherchannel

PPPoE sessions with active/standby Etherchannel support one-level or two-level hierarchical output policy-maps (with queueing settings) also support flat input policy-maps (without queueing settings). The policy-maps are configured using previously defined class maps. The traffic classes must be configured using the **class-map** command.

The output hierarchical policy-map and the input policy-map can be associated with the PPPoE sessions in one of the following ways:

- Configuration settings on a virtual template interface
- Dynamic configuration settings via external tools configured in the authentication, authorization, and accounting (AAA) model (for example, a radius server). For more information, see the *Intelligent Services Gateway Configuration Guide* and the *Cisco ASR 1000 Series Aggregation Services Routers Software Configuration Guide*.

The port-channel main interface must contain the following commands that create an active/standby scenario. Such a configuration will allow only a single interface to be active and forwarding traffic at any time.

- **interface port-channel1**
- **lacp fast-switchover**
- **lacp max-bundle 1**

How to Configure PPPoGEC Per Session QoS

Configuring QoS on PPPoE Sessions with Etherchannel Active/Standby

To configure QoS on PPPoE sessions, you must specify the virtual template to use for PPP sessions on the Etherchannel interface, specify the name of the service policy that is applied to input traffic, and specify the output traffic. This configuration shows how to associate the output hierarchical policy-map and the input policy-map with the PPPoE sessions by defining a virtual template interface.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **interface virtual-template** *number*
4. **service-policy output** *policy-map-name*
5. **service-policy input** *policy-map-name*
6. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:**	Enables privileged EXEC mode. • Enter your password if prompted.

	Command or Action	Purpose
	`Device> enable`	
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**interface virtual-template** *number* **Example:** `Device(config)# interface virtual-template 99`	Creates a virtual template interface that can be configured and applied dynamically in creating virtual access interfaces, and enters interface configuration mode. • Specify the virtual template to use for PPP sessions on the Etherchannel interface.
Step 4	**service-policy output** *policy-map-name* **Example:** `Device(config-if)# service-policy output session_parent`	Specifies the name of the service policy that is applied to output traffic.
Step 5	**service-policy input** *policy-map-name* **Example:** `Device(config-if)# service-policy input session_ingress`	Specifies the name of the service policy that is applied to input traffic.
Step 6	**end** **Example:** `Device(config-if)# end`	Exits interface configuration mode and returns to privileged EXEC mode.

Configuration Examples for PPPoGEC Per Session QoS

Example: QoS on PPPoE Sessions with Etherchannel Active/Standby

The following example shows the session_parent hierarchical policy-map and the session_ingress policy-map. These policy-maps are attached to a virtual template interface using the **service-policy** command.

```
policy-map session_child
  class voice
    priority level 1
    police cir 256000
    set precedence 5
  class web
    bandwidth remaining ratio 10
  class p2p
```

```
    bandwidth remaining ratio 1
    set precedence 1
  class class-default
    set precedence 2
    bandwidth remaining ratio 5
!
policy-map session_parent
  class class-default
    bandwidth remaining ratio 1
    shape average 25000000
    service-policy session_child
!
policy-map session_ingress
  class voip
    police cir 256000
  class p2p
    police cir 256000 pir 512000
      conform-action set-prec-transmit 1
      exceed  set-prec-transmit 0
      violate drop
  class class-default
    police cir 5000000
      conform-action set-prec-transmit 2
      exceed  drop
!
interface Virtual-template 99
 service-policy output session_parent
 service-policy input session_ingress
```

Additional References for PPPoGEC Per Session QoS

Related Documents

Related Topic	Document Title
Cisco IOS commands	*Cisco IOS Master Command List, All Releases*
QoS commands: complete command syntax, command modes, command history, defaults, usage guidelines, and examples	*Cisco IOS Quality of Service Solutions Command Reference*
Modular Quality of Service Command-Line Interface	"Applying QoS Features Using the MQC" module
Configuring RADIUS-based policing	*Intelligent Services Gateway Configuration Guide*
CISCO ASR 1000 Series software configuration	*Cisco ASR 1000 Series Aggregation Services Routers Software Configuration Guide*

Technical Assistance

Description	Link
The Cisco Support and Documentation website provides online resources to download documentation, software, and tools. Use these resources to install and configure the software and to troubleshoot and resolve technical issues with Cisco products and technologies. Access to most tools on the Cisco Support and Documentation website requires a Cisco.com user ID and password.	http://www.cisco.com/cisco/web/support/index.html

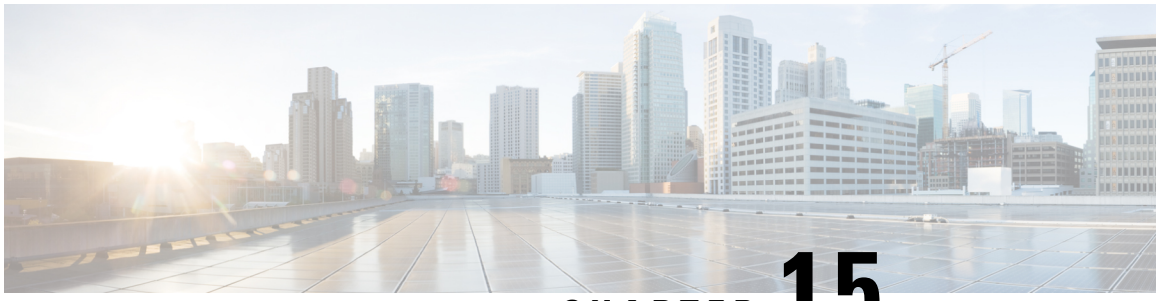
Feature Information for PPPoGEC Per Session QoS

The following table provides release information about the feature or features described in this module. This table lists only the software release that introduced support for a given feature in a given software release train. Unless noted otherwise, subsequent releases of that software release train also support that feature.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to www.cisco.com/go/cfn. An account on Cisco.com is not required.

Table 14: Feature Information for PPPoGEC Per Session QoS

Feature Name	Releases	Feature Information
PPPoGEC: Per Session QoS	Cisco IOS XE Release 3.7S	This feature supports the configuration of specific QoS policies on PPPoE sessions on the PTA, LAC, and LNS for broadband deployments. This feature was introduced on Cisco ASR 1000 Series Routers. In Cisco IOS XE Release 3.8S, support was added for per-session QoS in 1:1 mode for PPPoGEC. Also, support for Point-to-Point Protocol (PPP) and IP over PPPoE was also added for PPPoGEC. In Cisco IOS XE Release 3.9S, support was added for IP session over GEC in 1:1 mode.



CHAPTER 15

IPv6 Selective Packet Discard

The selective packet discard (SPD) mechanism manages the process level input queues on the RP. SPD provides priority to routing protocol packets and other important traffic control Layer 2 keepalives during periods of process level queue congestion

Finding Feature Information

Your software release may not support all the features documented in this module. For the latest caveats and feature information, see Bug Search Tool and the release notes for your platform and software release. To find information about the features documented in this module, and to see a list of the releases in which each feature is supported, see the feature information table.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to https://cfnng.cisco.com/. An account on Cisco.com is not required.

Information About IPv6 Selective Packet Discard

SPD in IPv6 Overview

The SPD mechanism manages the process level input queues on the RP. SPD provides priority to routing protocol packets and other important traffic control Layer 2 keepalives during periods of process level queue congestion.

SPD State Check

The SPD state check is performed on the IPv6 process input queue on the RP. High-priority packets, such as those of IP precedence 6, are not applied to SPD and are never dropped. All remaining packets, however, can

be dropped depending on the length of the IPv6 packet input queue and the SPD state. The possible SPD states are as follows:

- Normal: The process input queue is less than the SPD minimum threshold.
- Random drop: The process input queue is between the SPD minimum and maximum thresholds.
- Max: The process input queue is equal to the SPD maximum threshold.

The size of the process input queue governs the SPD state: normal (no drop), random drop, or max. When the process input queue is less than the SPD minimum threshold, SPD takes no action and enters normal state. In the normal state, no packets are dropped. When the input queue reaches the maximum threshold, SPD enters max state, in which normal priority packets are discarded. If the input queue is between the minimum and maximum thresholds, SPD enters the random drop state, in which normal packets may be dropped.

SPD Mode

Three IPv6 SPD modes are supported: none (which is the default), aggressive drop, and OSPF mode. The aggressive drop mode discards incorrectly formatted packets when the IPv6 is in the random drop state. OSPF mode provides a mechanism whereby OSPF packets are handled with SPD priority.

SPD Headroom

With SPD, the behavior of normal IPv6 packets is not changed. However, routing protocol packets are given higher priority, because SPD recognizes routing protocol packets by the IPv6 precedence field. Therefore, if the IPv6 precedence is set to 6, then the packet is given priority.

SPD prioritizes IPv6 packets with a precedence of 6 by allowing the Cisco IOS software to queue them into the process level input queue above the normal input queue limit. The number of packets allowed in excess of the normal limit is called the SPD headroom. The SPD headroom default is 100, which means that a high precedence packet is not dropped if the size of the input hold queue is lower than 175 (which is the input queue default size + SPD headroom size).

Because Interior Gateway Protocols (IGPs) and link stability are tenuous and crucial, such packets are given the highest priority and are given extended SPD headroom with a default of 10 packets. These packets are not dropped if the size of the input hold queue is lower than 185 (input queue default size + SPD headroom size + SPD extended headroom).

Non-IPv6 packets such as Connectionless Network Service Intermediate System-to-Intermediate System (CLNS IS-IS) packets, PPP packets, and High-Level Data Link Control (HDLC) keepalives are treated as normal priority as a result of being Layer 2 instead of Layer 3. In addition, IGPs operating at Layer 3 or higher are given priority over normal IPv6 packets, but are given the same priority as Border Gateway Protocol (BGP) packets. Therefore, during BGP convergence or during times of very high BGP activity, IGP hellos and keepalives often are dropped, causing IGP adjacencies to fail.

How to Configure IPv6 Selective Packet Discard

Configuring the SPD Process Input Queue

SUMMARY STEPS

1. **enable**

2. **configure terminal**
3. **ipv6 spd queue max-threshold** *value*
4. **ipv6 spd queue min-threshold** *value*
5. exit
6. **show ipv6 spd**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Router> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Router# configure terminal`	Enters global configuration mode.
Step 3	**ipv6 spd queue max-threshold** *value* **Example:** `Router(config)# ipv6 spd queue max-threshold 60000`	Configures the maximum number of packets in the SPD process input queue.
Step 4	**ipv6 spd queue min-threshold** *value* **Example:** `Router(config)# ipv6 spd queue max-threshold 4094`	Configures the minimum number of packets in the IPv6 SPD process input queue.
Step 5	exit **Example:** `Router(config)# exit`	Returns the router to privileged EXEC mode.
Step 6	**show ipv6 spd** **Example:** `Router# show ipv6 spd`	Displays IPv6 SPD configuration.

Configuring an SPD Mode

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **ipv6 spd mode** {**aggressive** | **tos protocol ospf**}

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Router> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Router# configure terminal`	Enters global configuration mode.
Step 3	**ipv6 spd mode** {**aggressive** | **tos protocol ospf**} **Example:** `Router(config)# ipv6 spf mode aggressive`	Configures an IPv6 SPD mode.

Configuring SPD Headroom

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **spd headroom** *size*
4. **spd extended-headroom** *size*
5. exit
6. **show ipv6 spd**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Router> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Router# configure terminal`	Enters global configuration mode.
Step 3	**spd headroom** *size* **Example:** `Router(config)# spd headroom 200`	Configures SPD headroom.

	Command or Action	Purpose
Step 4	**spd extended-headroom** *size* **Example:** `Router(config)# spd extended-headroom 11`	Configures extended SPD headroom.
Step 5	exit **Example:** `Router(config)# exit`	Returns the router to privileged EXEC mode.
Step 6	**show ipv6 spd** **Example:** `Router# show ipv6 spd`	Displays the IPv6 SPD configuration.

Configuration Examples for IPv6 Selective Packet Discard

Example: Configuring the SPD Process Input Queue

The following example shows the SPD process input queue configuration. The maximum process input queue threshold is 60,000, and the SPD state is normal. The headroom and extended headroom values are the default:

```
Router# ipv6 spd queue max-threshold 5000
Router# show ipv6 spd

Current mode: normal
Queue max threshold: 60000, Headroom: 100, Extended Headroom: 10
IPv6 packet queue: 0
```

Additional References

Related Documents

Related Topic	Document Title
IPv6 addressing and connectivity	*IPv6 Configuration Guide*
Cisco IOS commands	Master Commands List, All Releases
IPv6 commands	IPv6 Command Reference
Cisco IOS IPv6 features	IPv6 Feature Mapping

Standards and RFCs

Standard/RFC	Title
RFCs for IPv6	*IPv6 RFCs*

Technical Assistance

Description	Link
The Cisco Support and Documentation website provides online resources to download documentation, software, and tools. Use these resources to install and configure the software and to troubleshoot and resolve technical issues with Cisco products and technologies. Access to most tools on the Cisco Support and Documentation website requires a Cisco.com user ID and password.	http://www.cisco.com/cisco/web/support/index.html

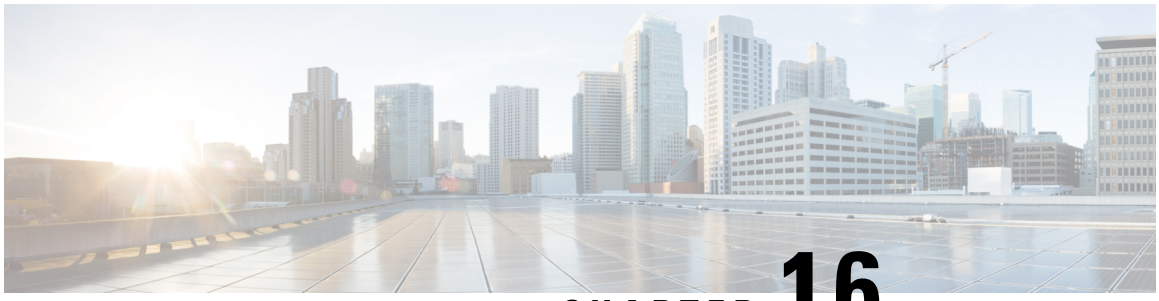
Feature Information for IPv6 Selective Packet Discard

The following table provides release information about the feature or features described in this module. This table lists only the software release that introduced support for a given feature in a given software release train. Unless noted otherwise, subsequent releases of that software release train also support that feature.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to www.cisco.com/go/cfn. An account on Cisco.com is not required.

Table 15: Feature Information for IPv6 Selective Packet Discard

Feature Name	Releases	Feature Information
IPv6: Full Selective Packet Discard Support	Cisco IOS XE Release 2.6	The SPD mechanism manages the process level input queues on the RP. SPD provides priority to routing protocol packets and other important traffic control Layer 2 keepalives during periods of process level queue congestion. The following commands were introduced or modified: **clear ipv6 spd**, **debug ipv6 spd**, **ipv6 spd mode**, **ipv6 spd queue max-threshold**, **ipv6 spd queue min-threshold**, **monitor event-trace ipv6 spd**, **show ipv6 spd**, **spd extended-headroom**, **spd headroom**.



CHAPTER 16

Per ACE QoS Statistics

The Per ACE QoS Statistics feature extends the QoS Packet Matching Statistics feature to allow you to track the number of packets and bytes matching individual access control elements (ACEs) used in a filter. The filter is part of the class-map definition of a quality of service (QoS) policy-map.

You can use the **show access-lists** command to display per-ACE statistics.

See the "QoS Packet Matching Statistics" module for information on defining a QoS packet filter and displaying the number of packets and bytes matching that filter.

Finding Feature Information

Your software release may not support all the features documented in this module. For the latest caveats and feature information, see Bug Search Tool and the release notes for your platform and software release. To find information about the features documented in this module, and to see a list of the releases in which each feature is supported, see the feature information table.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to https://cfnng.cisco.com/. An account on Cisco.com is not required.

Prerequisites for Per ACE QoS Statistics

Before you configure the **platform qos match-statistics per-ace** command to enable QoS per-ACE packet-matching statistics, you must configure the **platform qos match-statistics per-filter** command to enable QoS per-filter packet-matching statistics. If you do not, the CLI rejects the command and displays an error message.

Restrictions for Per ACE QoS Statistics

If a QoS policy-map is attached to the device when you configure the **platform qos match-statistics per-ace** command, the command does not take effect until you do one of the following:

- Reload the device.
- Detach all QoS policies and configure the command again.

Enabling the Per ACE QoS Statistics feature may increase CPU utilization on a scaled configuration. Before you enable it, you should weigh the benefits of the statistics information against the increased CPU utilization on the system.

Note You must configure the **platform qos match-statistics per-filter** command before you configure the **platform qos match-statistics per-ace** command.

Information About Per ACE QoS Statistics

Per ACE QoS Statistics Overview

The Per ACE QoS Statistics feature provides hit counters for ACEs used in QoS policies. When enabled, the feature adds QoS hit counters for any ACEs used in a QoS policy to the existing security access-list counters for that ACE. You can use the **show ip access-lists** command to display the access-list counters, as shown in this example:

```
Device# show ip access-lists

Extended IP access list A1
10 permit ip 10.1.1.0 0.0.0.255 any (129580275 matches)
Extended IP access list A6and7
10 permit ip 10.1.6.0 0.0.0.255 any (341426749 matches)
20 permit ip 10.1.7.0 0.0.0.255 any (398245767 matches)
Extended IP access list source
10 permit ip any host 10.1.1.5 (16147976 matches)
```

The QoS hit counters (for ACEs used in QoS policies) will be added to the counters shown in the sample output.

Note the following conditions when you enable the Per ACE QoS Statistics feature:

- The **show ip access-lists** command does not display interface information. This means that access-list counts are not interface-specific; they are aggregate counters of all hits for all features that use the ACEs and support the counts across all interfaces and directions.
- You can use the **show policy-map interface** command to display interface-specific counts if QoS per-filter packet matching statistics is enabled. However, this command displays only counts per-filter [access-control list (ACL) or access-group], not counts per-ACE, as shown in this example:

  ```
  Device# show policy-map interface GigabitEthernet0/0/2
  ```

```
 GigabitEthernet0/0/2

  Service-policy input: test-match-types

    Class-map: A1orA2-class (match-any)
      482103366 packets, 59780817384 bytes
      5 minute offered rate 6702000 bps
      Match: access-group name A1
        62125633 packets, 7703578368 bytes
        5 minute rate 837000 bps
      Match: access-group name A2
        419977732 packets, 52077238892 bytes
        5 minute rate 5865000 bps
```

- If an ACE is present in a QoS filter (that is, a match statement within a class map) but the packet does not match the ACE, the ACE counter is not incremented for that packet. This can happen in the following circumstances:
  - The ACE is used in a “deny” statement.
  - Other matching criteria in a “match-all” class-map definition (for example, “match ip prec 1”) prevent the packet from matching the class.
  - Other matching criteria in a “match-any” class-map definition (for example, “match ip prec 1”) match the packet and prevent it from matching the ACE match criteria (that filter precedes the ACE filter and the packet matches both statements).

- Access-list counts are an aggregate, for a particular ACE, of the hit counts for all features that use that ACE and support per-ACE counts. This means that a single packet might hit, and be counted by, multiple features using the same ACE, and, therefore, result in multiple counts for the same packet as it traverses each feature.

  The following example shows these multiple counts. Only 1,000 packets were received on the interface but the access-list counts show 2,000 hits, 1,000 for the security access list and 1,000 for the QoS service policy.

```
Device(config)# ip access-list extended A1
permit ip 32.1.1.0 0.0.0.255 any
class-map match-all A1-class
match access-group name A1
interface GigabitEthernet0/0/2
ip address 10.0.0.1 240.0.0.0
ip access-group A1 in
duplex auto
speed auto
media-type rj45
no negotiation auto
service-policy input simple
end

Device# show access-lists

Extended IP access list A1
10 permit ip 10.1.1.0 0.0.0.255 any (2000 matches)

Device# show policy-map interface GigabitEthernet0/0/2

Service-policy input: simple
Class-map: A1-class (match-all)
1000 packets, 124000 bytes
5 minute offered rate 4000 bps
```

```
Match: access-group name A1
Class-map: class-default (match-any)
0 packets, 0 bytes
5 minute offered rate 261000 bps, drop rate 0 bps
Match: any
```

How to Configure Per ACE QoS Statistics

Configuring Per ACE QoS Statistics

Before you begin

The **platform qos match-statistics per-filter** command must be configured to enable QoS per-filter packet-matching statistics. You can use the **show platform hardware qfp active feature qos config global** command to verify the status of packet-matching statistics.

```
Device# show platform hardware qfp active feature qos config global

Marker statistics are: disabled
Match per-filter statistics are: enabled <<<<<<<
Match per-ace statistics are: disabled <<<<<<
Performance-Monitor statistics are: disabled
```

SUMMARY STEPS

1. **platform qos match-statistics per-filter**
2. **platform qos match-statistics per-ace**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**platform qos match-statistics per-filter** **Example:** `Device(config)# platform qos match-statistics per-filter`	Enables QoS packet-matching statistics for individual filters in a class map.
Step 2	**platform qos match-statistics per-ace** **Example:** `Device(config)# platform qos match-statistics per-ace`	Enables QoS packet-matching statistics for ACEs used in QoS filters.

Additional References for Per ACE QoS Statistics

Related Documents

Related Topic	Document Title
Cisco IOS commands	*Cisco IOS Master Command List, All Releases*
QoS commands	*Cisco IOS Quality of Service Solutions Command Reference*
Defining a QoS packet filter and displaying the number of packets and bytes matching it	"QoS Packet Matching Statistics"

Technical Assistance

Description	Link
The Cisco Support and Documentation website provides online resources to download documentation, software, and tools. Use these resources to install and configure the software and to troubleshoot and resolve technical issues with Cisco products and technologies. Access to most tools on the Cisco Support and Documentation website requires a Cisco.com user ID and password.	http://www.cisco.com/cisco/web/support/index.html

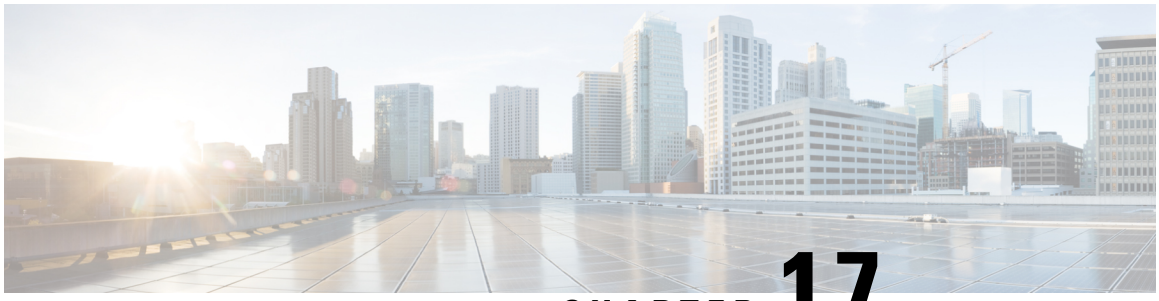
Feature Information for Per ACE QoS Statistics

The following table provides release information about the feature or features described in this module. This table lists only the software release that introduced support for a given feature in a given software release train. Unless noted otherwise, subsequent releases of that software release train also support that feature.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to www.cisco.com/go/cfn. An account on Cisco.com is not required.

Table 16: Feature Information for Per ACE QoS Statistics

Feature Name	Releases	Feature Information
Per ACE QoS Statistics	Cisco IOS XE Release 3.10S	Allows you to configure per ACE QoS statistics to track the number of packets and bytes matching individual ACEs used in a filter within a QoS service policy. The following command was introduced or modified: **platform qos match-statistics per-ace**.



CHAPTER 17

QoS Packet Policing

Traffic policing allows you determine whether network traffic is above or below a predetermined rate and to provide different treatment for such traffic. In its simplest form a policer (rate limiter) drops any traffic that exceeds a predetermined rate.

About QoS Policing

Why Traffic Policing

Allowing you to control the maximum rate of traffic transmitted or received on an interface, traffic policing is typically configured on interfaces at the edge of a network to limit traffic into the network. In most traffic policing configurations, traffic that falls within the rate parameters is transmitted whereas traffic that exceeds the parameters is dropped or marked (and transmitted).

Note Unlike a shaper, a policer does not buffer packets. Rather, the specified action is taken immediately.

Typically, we use policers for admission control: queue or network.

Queue Admission Control limits the amount of data that can enter a queue. A *priority queue* is representative of this category wherein we avoid latency by limiting the rate at which packets may be enqueued.

Network Admission Control enforces a contract between the network administrator (service provider) and his customers. Generally both will agree on the rate at which the provider should accept traffic. This could be the

service-rate (max rate for all the traffic customer sends to provider) or a *per-class restriction* (e.g., the amount of priority traffic a customer may send).

- Using network admission control, you may decide to either drop excess traffic immediately or mark that traffic as 'out of contract.' If the latter, you can either provide that traffic a lesser treatment or drop it first if (and when) congestion occurs within this network.

Policer Definitions

Note The terms *Policer* and *Rate Limiter* usually refer to the same QoS mechanism. Policer (*Policing*) will be used throughout this document.

A policer is a device that allows you to define different treatments for packets within the same traffic class depending on whether packets are received above or below a specified rate(s).

In its simplest form, a policer indicates that traffic above a specified rate should be dropped:

```
policy-map police-all-traffic
  class class-default
    police 1m
```

Traffic through this class arriving at a rate less than 1 Mbps is considered *conforming* (adhering to the specified rate). The default action for conforming traffic is to forward packets.

Traffic arriving at a rate exceeding 1 Mbps is considered *exceeding* the configured rate. The default action for exceeding traffic is to drop packets.

The following definitions are relevant to understanding the sections that follow.

Policer Actions

In the previous example, copied below, we used a policer in its most basic form:

```
policy-map police-all-traffic
  class class-default
    police 1m
```

Conforming traffic was allowed to pass through the policer (transmitted traffic below 1m) whereas exceeding traffic was dropped. We took *immediate action* when we recognized that traffic had exceeded the specified rate. However, you may not want to always take immediate action. You might want to *defer action* rather than immediately drop traffic.

For example, you may decide that traffic above the predetermined rate should only be dropped if the network is congested. If so, you might choose to forward all traffic but mark something in the packet (e.g., DSCP) differently for conforming and exceeding traffic. The decision on whether or not to drop can then be made at the congestion point.

In the following example, we mark rather than drop traffic. We define a traffic class as any traffic arriving with a DSCP value of AF41 and demote traffic exceeding a specified rate to AF42:

```
policy-map ma
rk-out-of-contract
  class AF41
```

```
police 1m conform-action transmit exceed-action set-dscp-transmit AF42
```

The *conform-action* is to transmit traffic (simply forward, default action) arriving at a rate less than or equal to the specified 1 Mbps rate.

The *exceed-action* for traffic exceeding 1 Mbps is to mark the packet's DSCP value rather than drop traffic.

Transmit and drop represent *actions* specified for traffic *conforming* to or *exceeding* the specified rate. You specify an action with the **police** command. Supported actions are listed in the following table.

Note The rules for policer actions are very similar to those for the **set** command.

You can only mark Layer 2 and outer Layer 3 headers.

Multi-Action Policer

In the previous section we saw how a policer can be configured to mark some field in the packet. In fact, we can mark multiple fields in the packet.

You can apply multiple actions to traffic within each rate designation, analogous to how you configure multiple set actions within a traffic class. For example, if you know a packet will be transmitted through both a TCP/IP and a Frame Relay environment, you can change the DSCP value of the exceeding or violating packet, and also set the Frame Relay Discard Eligibility (DE) bit from 0 to 1 to indicate lower priority.

When specifying multiple policing actions, observe the following:

- You must enter policy-map class police configuration (config-pmap-c-police) submode.
- You can specify a maximum of four actions simultaneously, one line per action.
- You cannot specify contradictory actions such as **conform-action transmit** and **conform-action drop**.

Analogous to the **set** command, you can either configure multiple actions on the same packet (e.g., marking Layer 2 and Layer 3 fields) or define actions for different traffic types (e.g., marking the DSCP value in IPv4 packets and experimental (EXP) bits in MPLS packets).

In the following example, we cap RTP traffic (`rtp-traffic`) at 1 Mbps and drop traffic exceeding that rate (`exceed-action drop`). For conforming traffic, we mark both the COS and DSCP values in IPv4 packets and the COS and EXP bits in MPLS packets:

```
class rtp-traffic
  police cir 1000000
    conform-action set-cos-transmit 4
    conform-action set-dscp-transmit af41
    conform-action set-mpls-exp-topmost-transmit 4
    exceed-action drop
```

All packets in a traffic class count towards the rate seen by that class but actions are applied only to applicable traffic. For example, imagine that IPv4 and MPLS packets are classified into the same traffic class and a policer is configured to mark a specific DSCP value. Both IPv4 and MPLS packets count towards the observed rate, but only IPv4 packets can be marked.

Note Configuring multiple actions is supported for single and dual-rate policers. (See Single-Rate, Two-Color Policer, on page 263 and Dual-Rate, Three-Color Policer, on page 266.)

A Note on CLI Variants

This section shows how multiple variants of the CLI can achieve the same result.

Context

The variations have emerged in different Cisco IOS software releases over time and as software trains have merged. Within the same software release, three equivalent variants exist. To avoid backwards compatibility issues, we decided to retain the variants. Please note, however, that the software implementing the policing is identical regardless of the CLI variant used.

Illustration

For the following examples, we set **police** to 10 Mbps, **conform action** to transmit (default), and **exceed action** to drop (default). At a "high" level we have three variants of the **police** command that achieve the same result: **police** *value*, **police cir** *value*, and **police rate** *value*. This set of variants is equivalent to: **police [cir|rate]***value*, where **cir** and **rate** are optional. With a rate of 10 Mbps, we can build the following command: **police [cir|rate] 10m**.

Using each variant to configure policing:

```
policy-map policer-cli-example
  class class-default
    police 10000000


policy-map policer-cli-example
  class class-default
    police cir 10m


policy-map policer-cli-example
  class class-default
    police rate 10m
```

To verify that the three variants yield the same result, you can use two stages of verification:

1. Issue **show policy-map interface** to display the configuration within IOS.
2. Issue **show platform hardware qfp active feature qos interface** to illustrate how we program hardware. This display is unchanged regardless of the CLI variant used.

Let's run Step 1:

```
show policy-map int GigabitEthernet1/0/0

  Service-policy input: policer-cli-example

    Class-map: class-default (match-any)
      162 packets, 9720 bytes
      5 minute offered rate 2000 bps, drop rate 0000 bps
      Match: any
```

```
      police:
          cir 10000000 bps, bc 312500 bytes
        conformed 212 packets, 12720 bytes; actions:
          transmit
        exceeded 0 packets, 0 bytes; actions:
          drop
        conformed 2000 bps, exceeded 0000 bps
```

Next, let's run Step 2:

```
show platform hardware qfp active feature qos int g1/0/0
```

```
Interface: GigabitEthernet1/0/0, QFP interface: 12
  Direction: Input
    Hierarchy level: 0
    Policy name: policer-cli-example
      Class name: class-default, Policy name: policer-cli-example
        Police:
            cir:  10000000 bps,  bc: 315392 bytes
            pir: 0 bps,  be: 315392 bytes
            rate mode: Single Rate Mode
            conformed: 16 packets, 960 bytes; actions:
              transmit
            exceeded: 0 packets, 0 bytes; actions:
              drop
            violated: 0 packets, 0 bytes; actions:
              drop
            color aware: No
            green_qos_group: 0, yellow_qos_group: 0
```

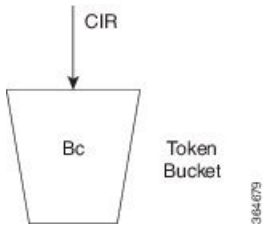
Single-Rate, Two-Color Policer

A single-rate, two-color policer (1R2C) determines whether traffic is above or below a predetermined rate (CIR in bps) and allows you to take action in either instance. The possible actions for any arriving packet are conform (packet counts as traffic falling below the CIR) and exceed (packet counts as traffic exceeding the CIR).

We need to allow for any *potential burstiness*. This behavior occurs when many packets arrive together, and the arrival rate over a short interval exceeds the CIR while the arrival rate over a longer range might conform to the CIR. To accommodate bursts yet enforce our predetermined CIR over time, we use a *token bucket* scheme.

Applying this scheme, we can represent a single-rate, two-color policer with a single-token bucket:

Figure 64: Single-Rate, Two-Color Policer



Tokens are continuously replenished at CIR and the depth of the bucket is Bc. If the bucket is full, additional tokens arriving are lost.

When a packet arrives, the policer assesses whether the bucket contains enough tokens (bytes) to *cover that incoming packet* (sufficient bytes to match the packet length). If so, the packet is regarded as conforming, the action is taken and the appropriate number of tokens (packet length) is removed from the bucket.

If the packet arrives and the bucket contains insufficient tokens to cover the packet, the exceed action is taken; the number of tokens in the bucket are unchanged. Subsequent packets may find that the bucket has replenished sufficiently to be now designated "conforming." If no packets arrive, the bucket continues to fill to the burst limit (Bc).

Specifying the bucket depth determines the allowable amount of burstiness for conforming traffic (how many bytes/packets) that may arrive closely together, assuming the bucket has had time to refill.

In this example we have specified a CIR of 10 Mbps and a burst allowance of 15000 bytes. So, a burst of 10 MTU-sized packets on an Ethernet interface could be designated conforming:

```
policy-map police-with-burst
  class class-default
    police cir 10m bc 15000
```

Note The current IOS CLI enables you to configure policing in multiple ways yet accomplish the same result. See the section A Note on CLI Variants, on page 262.

Single-Rate, Three-Color Policer

A single-rate, three-color policer (1R3C) supports three possible output states: conform, exceed and violate. The definition of conform is analogous to that in a 1R2C policer – traffic that adheres to a predetermined rate allowing for some burst tolerance.

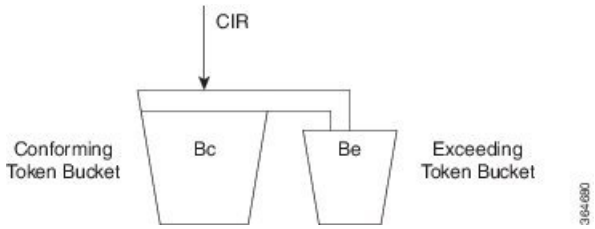
The difference stems from how we designate traffic that does not conform – traffic that a two-color policer would designate as exceed. We introduce further granularity where this traffic could be exceed or violate. Essentially, traffic that bursts ‘minimally’ above the CIR is designated as exceed but more sustained bursts above the CIR would be designated as violate.

To achieve this behavior we introduce a second token bucket. Just as the conforming token bucket is used to differentiate between traffic that conforms or exceeds, the excess token bucket enables us to differentiate between traffic that exceeds or violates.

Here are the bucket scenarios:

- The conforming token bucket is initially full (the number of bytes specified as Bc (conforming burst size).
- The exceeding token bucket is initially full (the number of bytes specified in the Be (excess burst size).
- If the conforming token bucket is full when tokens arrive (at the CIR, analogous to a 1R2C policer), they overflow into the excess token bucket.
- If both buckets are full, further tokens are lost.

Figure 65: Single-Rate, Three-Color Policer



The conforming bucket here behaves as it does in the 1R2C scenario. If the bucket contains sufficient tokens to cover the incoming packet, the packet is considered "conforming," the conforming action occurs and we remove an appropriate number of tokens from the bucket. The exceeding bucket is unaffected and we continue to replenish the conforming bucket (Bc) at CIR.

However, if the conforming bucket is full and additional tokens arrive they are not immediately lost. Instead, they overflow into the exceeding bucket. If this bucket is full, excess tokens are lost.

Similarly, when a packet arrives and the conforming bucket has insufficient tokens to cover that packet we cannot immediately declare it as exceeding; it might be exceeding or violating. If the exceeding bucket has enough tokens to cover the packet, the exceeding action is taken, and we remove the necessary number of tokens from the exceeding bucket. No bytes are removed from the conforming bucket.

If neither bucket, conforming or exceeding, has enough tokens to cover the packet, it is categorized as violating and the appropriate action is taken. Neither the conforming nor exceeding bucket is decremented:

If neither bucket, conforming or exceeding, has enough tokens to cover the packet, it is categorized as violating and the appropriate action is taken. Neither the conforming nor exceeding bucket is decremented:

```
policy-map ingress-enforcement
  class af41-metering
    police cir percent 10 bc 5 ms be 10 ms
    conform-action set-dscp-transmit af41
    exceed-action set-dscp-transmit af42
    violate-action drop
```

In this example we are policing traffic (for class af41) to 10% of the interface's bandwidth and the following apply:

- Traffic (**`conform-action set-dscp-transmit af41`**) burst up to 5 ms is forwarded and still marked as af41.
- Traffic (**`exceed-action set-dscp-transmit af42`**) burst exceeding 5 ms and up to an additional 10ms of burst is marked as af42. Elsewhere in the network, when we detect af42, we know it was received beyond the agreed contract [at the edge of the network]; under congestion, we could drop it first.
- Traffic (**`violate-action drop`**) burst beyond 15 ms above our CIR is considered violating and dropped immediately.

Note We only replenish the exceeding bucket when the conforming bucket is full. So, if you send a non-bursty stream at a rate exceeding the CIR, shortly, both the conforming and exceeding buckets will be drained; we do not replenish the exceeding bucket. All subsequent packets are considered either conforming or violating.

Dual-Rate, Three-Color Policer

Traffic rates are easier to understand than traffic burstiness. When specifying a contract for network admission control (See Why Traffic Policing, on page 259), you might have trouble describing expectations in terms of multiple burst sizes above a single rate. The dual-rate, three-color (2R3C) policer simplifies matters by primarily employing rates to differentiate conform, exceed and violate. It also introduces a second rate, PIR (Peak Information Rate)

CIR and PIR have the following characteristics:

- Traffic below the CIR is conforming.
- Traffic greater than CIR but less than PIR is exceeding.
- Traffic above PIR is violating.

You specify these rates with the **cir** and **pir** keywords of the **police** command. (For details, please refer to the command page for police, on page 285.)

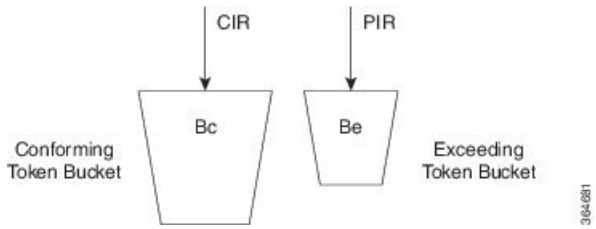
With a 2R3C policer, unlike a 1R3C, we replenish token buckets independently whenever a packet arrives at the policer. We refill conforming buckets at rate CIR; it can contain up to value Bc. ; exceeding buckets, at PIR; it can contain up to value Be.

Note PIR must exceed CIR and overflow between buckets is disallowed.

If a steady stream of packets arrives at a rate exceeding the CIR but less than the PIR, all packets are marked either conforming or exceeding. With the 1R3C policer, this scenario would have resulted in marking a minimal number of packets as exceeding and a majority as conforming or violating.

A 2R3C policer supports three possible actions for each packet: conform, exceed, and violate. Traffic entering the interface configured with a dual-rate policer is placed into one of these action categories, which dictates how we treat a packet. For instance, in the most common configuration, you can configure to send packets that either conform or exceed (with a decreased priority), and to drop packets that violate.

Figure 66: Dual-Rate, Three-Color Policer



When a packet arrives, we assess whether ample tokens exist in the conforming and exceeding buckets to cover that packet. If so, we take the conforming action (typically, transmit or transmit and mark) and remove the necessary tokens to transmit the packet from both buckets.

If the Exceeding Token Bucket (but not the Conforming Token Bucket) contains sufficient tokens to cover the packet, we take the exceeding action (typically, transmit or transmit and marking). The appropriate number of tokens are removed from the exceeding bucket only.

If neither bucket has sufficient tokens to cover the packet, the violating action is taken (typically, transmit, transmit and marking, or drop):

```
policy-map ingress-enforcement
  class af41-metering
    police cir 100k bc 3000 pir 150k be 3000 conform-action set-dscp-transmit af41
exceed-action set-dscp-transmit af42 violate-action drop
```

Observe how code from the preceding example and the corresponding code from Single-Rate, Three-Color Policer, on page 264 differ:

```
cir 100k bc 3000 pir 150k be 3000
cir percent 10 bc 5 ms be 10 ms
```

In the immediate example, we handle traffic accordingly:

- Up to 100Kbps (allowing for bursts up to 3,000 bytes) as conforming and forward it with DSCP marked as af41.
- Above 100Kbps but less than 150Kbps (again allowing a 3,000 byte burst) as exceeding and forward it marked as af42.
- Above 150Kbs as violating; we drop it.

Configuring Rates and Burst Parameters

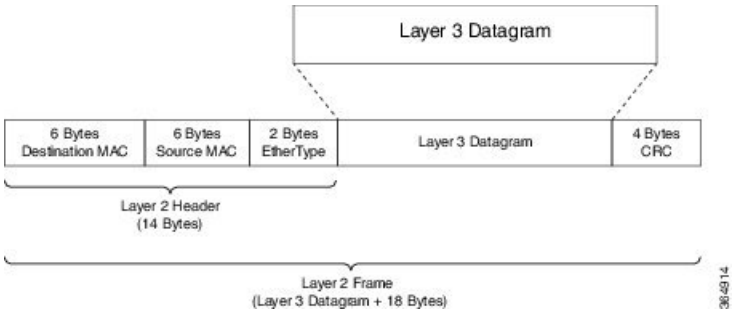
What's Included in the Policer-Rate Calculation (Overhead Accounting)

When specifying a rate or burst value, you should know how the policer assesses a packet's length (subsequently referred to as the *policing length*) when you evaluate conformance to those values. Briefly, a policer includes the Layer 3 datagram Layer 2 header lengths but neither CRC nor inter-packet overhead.

To further illustrate, consider an IP datagram transported over a GigabitEthernet link.

Layer 3 Datagram

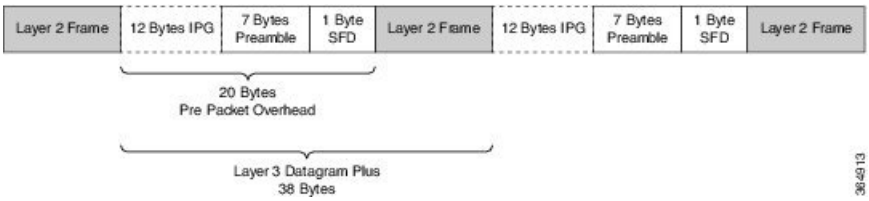
First, we encapsulate it in an Ethernet frame, which adds 14 bytes of Layer 2 header and an additional 4 bytes of CRC to each datagram (18 bytes):



Ethernet Overhead

To transmit this frame over the physical medium, Ethernet requires a minimum inter-packet gap equivalent to a transmit time for 12 bytes of data. After the gap, we require seven bytes of preamble followed by a single

byte start-of-frame delimiter (SFD) (Ethernet inter-packet overhead = 12 bytes IPG + 7 bytes Preamble + 1 byte SFD = 20 bytes).

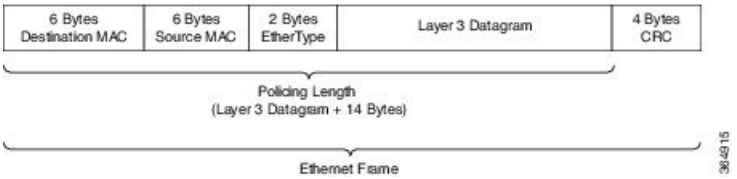


So, if you send multiple Ethernet frames sequentially, the per-packet overhead for each Layer 3 datagram is an additional 38 bytes (encapsulation [18 bytes] + Ethernet inter-packet overhead [20 bytes]). For example, if you sent 100 byte IP datagrams at line rate on a GigabitEthernet link, and used the following formula, the expected throughput in packets per second would be:

```
Linerate / Bits Per
  Byte / ( Layer 3 length + Per Packet Overhead) = Packets Per Second

1 Gbps / 8 / ( 100 + 38 ) = 905, 797 pps
```

From the policer's perspective, the packet's length is the Layer 3 datagram + Layer 2 header length (14 bytes on a GigabitEthernet interface):

Policing Length



Now consider a 500 Mbps policer configured on a GigabitEthernet interface. As in the previous example, we will send all 100 byte IP datagrams to the policer, resulting in a policing length of 100 byte datagram length + the 14 byte (Ethernet Layer 2 header). According to the following formula, the anticipated throughput would now be:

```
Policer Rate / Bits
  per Byte / (Layer 3 length + Layer 2 header length) = Packets Per Second

500 Mbps / 8 / (100 + 14) = 548,246 pps
```

Note Packets marked as *conforming* by a 500 Mbps policer will consume considerably more than 500 Mbps of physical bandwidth!

Policer on Logical Interface

On egress, a policer is unaware of the final physical interface type (tunnels can move between interfaces) and therefore the policer is unaware of the final Layer 2 overhead. So, the latter is excluded from the policing length. Similarly, because the policer cannot predict the extent of packet expansion due to overhead, if we configure encryption, we will not include encryption overhead in policer rate calculations. The egress policer will include the Layer 3 datagram and any tunnel headers (e.g., additional IP header, GRE header).

On ingress, because a policer is aware of the receiving interface type, policing on a tunnel interface includes Layer 2 overhead plus any tunnel headers.

The following table illustrates the dependencies of policer rate calculations on a ASR 1000 Series Aggregation Services Router. Be aware that we present only a subset of all permutations:

where the values are defined as follows:

- 0 - svti ('tunnel mode ipsec ipv4') has no overhead
- 14 - Layer 2 Ethernet header size
- 20 - the IP/IP header size
- 24 - the IP/GRE header size (20 + 4)

Policer on ATM Interfaces

If a policer is configured on an ATM interface, the policing length includes the Layer 3 datagram and the ATM adaption layer (AAL) header. For AAL5SNAP encapsulation length, this means that we include eight bytes of header in the policing length; for AAL5NLPID encapsulation, two bytes.

This calculation differs sharply from that applied to scheduling, where we include the complete *AAL PDU* and *cell tax*.

Changing What's Included - Overhead Accounting Adjustment

In prior sections, we described what is included by default in policer rate calculations. But what happens when you want to deviate from the default? For example, what if you want to express CIR as the physical bandwidth that would be consumed on a link? For an Ethernet interface you would include the 4 byte CRC and the 20 bytes inter-packet overhead required per packet.

Alternatively, you (a service provider) might want to police customers' traffic at Layer 3 rates. Because datagram length is unchanged as a packet traverses different interface types (or encapsulating protocols), we would not include Layer 2 header length in policer rate calculations.

Note Any interface that supports QoS policies will support overhead accounting adjustment.

Note Changing overhead accounting may impact the network. For example, if you use a policer for network admission control, you might need to configure a corresponding shaper on the equipment that connects to that network. The two views of what is included in CIR (shaper and policer) should match.

In the following example we want to include all inter-packet overhead such that a policer will allow up to 50% of the traffic on the physical link to be conforming. By adding 24 bytes per packet (**user-defined 24**) we address the 4 byte CRC and the 20-byte inter-packet overhead.

```
policy-map ethernet-physical-example
  class class-default
    police cir percent 50 account user-defined 24
```

Using the **atm** keyword of the **police account** command, you can direct the policer to compensate for ATM cell division and cell padding (ATM cell tax) in rate calculations.

To include cell tax and cover the AAl5 trailer, a router first adds 8 bytes to the policing length. Then, it calculates the number of ATM cells (48 bytes of data carried per 53 byte cell) required to carry the packet and multiplies this number by 53. For example, a 46 byte datagram would require 2 cells and therefore, if cell tax is included, the policing length would be considered "106 bytes."

In the following example, we show a 5 Mbps policer, which must include the cell-tax in its rate calculations:

```
policy-map include-cell-tax-example
  class class-default
    police cir 5000000 account user-defined 0 atm
```

The **atm** in the configuration dictates that we include the cell tax.

Restrictions for Overhead Accounting Adjustment

- If you enable overhead accounting on a child policy, then you must enable overhead accounting on the parent policy.
- In a policy-map, you must either enable or disable overhead accounting for all classes in a policy. Within the same policy, you cannot enable overhead accounting for some classes and disable overhead accounting for other classes.
- Overhead accounting is not reflected in any QoS counters (e.g., classification, policing, or queuing).
- You can enable overhead accounting on top-level parent policies as well as on both middle-level and bottom-level child policies. Child policies inherit overhead accounting policies configured at the "parent" or "grandparent" level.
- The overhead accounting type or value used within a policy-map and between the parent and the child policy-maps (in a hierarchical policy-map structure) must be consistent.

Overhead Accounting Adjustment (Predefined Options)

Through some predefined CLI options (based on broadband use cases), you can specify the encapsulation while the router adds or subtracts the appropriate number of bytes (see the following table).

Imagine that we send (or receive) traffic on an Ethernet interface to a DSLAM (digital subscriber line access multiplexer) elsewhere in the network. Although we are encapsulating in Ethernet frames (e.g., Dot1Q or Q-in-Q), the DSLAM encapsulates in some form of ATM encapsulation. We want the policer to execute on traffic as it would appear after the DSLAM. In all instances, we would add cell-tax to the policing length.

In the following example, we apply predefined overhead accounting values. If we receive Dot1Q-encapsulated packets on an Ethernet interface, an upstream DSLAM receives *AAL5-Mux 1483 routed encapsulated* packets, then strips the ATM and adds the Ethernet headers. On the ATM interface, the datagram would have 3 bytes of additional AAL headers but would not have the 18 bytes of Ethernet headers (including Dot1Q). So, the PDU would be 15 bytes less on the ATM interface (we subtract 15 bytes from the policing length and then add the cell-tax):

```
policy-map atm-example
  class class-default
    police 5000000 account dot1q aal5 mux-1483routed
```

Default Burst Sizes

If you don't explicitly configure a burst tolerance value (Bc or Be), IOS will configure a default. This default burst tolerance is 250 ms of data based on the appropriate rate. For example, if the CIR is 100 Mbps then 250 mS of this rate would be 100000000/8 x 0.250 = 3125000 bytes.

Bc and Be for a single rate policer are always based on the CIR. The Be for a dual-rate policer is based on the PIR:

Note When configuring a policer for queue admission control (See Why Traffic Policing, on page 259), set Bc to something suitable for applications in that queue (e.g., for a voice application, set Bc to 10 milliseconds or less).

```
policy-map policer-default
  class af41
    police cir 20000000 pir 40000000 conform-action transmit  exceed-action
                \ set-dscp-transmit af42 violate-action set-dscp-transmit af43

 show policy-map interface
 GigabitEthernet1/0/0

  Service-policy input: policer-default

    Class-map: af41 (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0000 bps, drop rate 0000 bps
      Match:  dscp af41 (34)
      police:
          cir 20000000 bps, bc 625000 bytes           1
          pir 40000000 bps, be 1250000 bytes          2
        conformed 0 packets, 0 bytes; actions:
          transmit
        exceeded 0 packets, 0 bytes; actions:
          set-dscp-transmit af42
        violated 0 packets, 0 bytes; actions:
          set-dscp-transmit af43
        conformed 0000 bps, exceeded 0000 bps, violated 0000 bps
```

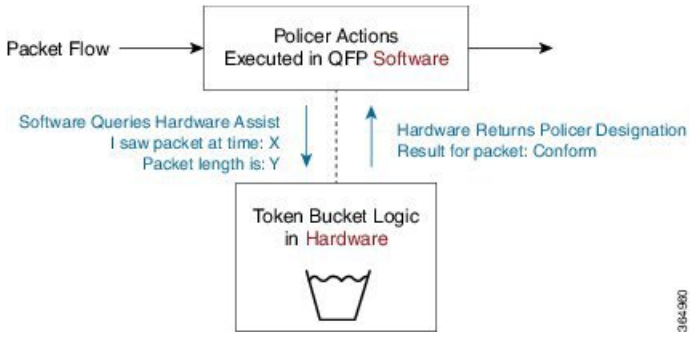
Note The dual-rate policer as well as Bc and Be default to 250ms based on the CIR (**1**) and PIR (**2**), respectively.

Rate and Burst Sizes Programmed in Hardware

On the Cisco ASR 1000 router platform, policer rate calculations are performed in *dedicated hardware*.

While *hardware assist* enables you to scale the number of policers independent of performance impact, it imposes some restrictions on the programmable rate and burst value combinations.

Figure 67:



Consider a simple policy with a single-rate, two-color policer:

```
policy-map hardware-example
  class class-default
    police cir 1m bc 3000
```

Output from the **show policy-map interface** command confirms that IOS has accepted the configured CIR and Bc values:

```
show policy-map interface g1/0/0


GigabitEthernet1/0/0

  Service-policy input: hardware-example

    Class-map: class-default (match-any)
      337 packets, 167152 bytes
      5 minute offered rate 2000 bps, drop rate 0000 bps
      Match: any
      police:
          cir 1000000 bps, bc 3000 bytes                    *
        conformed 337 packets, 167152 bytes; actions:
          transmit
        exceeded 0 packets, 0 bytes; actions:
          drop
        conformed 2000 bps, exceeded 0000 bps
```

***** CIR and Bc configured as expected

If you look at the dataplane, however, you can see the values actually programmed in hardware. Following is the output of the **show platform qfp active feature qos interface** command, which displays the actual policer values in hardware:

```
show platform hardware qfp active feature qos interface gig1/0/0


Interface: GigabitEthernet1/0/0, QFP interface: 9

  Direction: Input
    Hierarchy level: 0
    Policy name: hardware-example
      Class name: class-default, Policy name: hardware-example
        Police:
            cir:  1000000 bps,  bc: 3264 bytes              *
            pir: 0 bps,  be: 3008 bytes
```

```
rate mode: Single Rate Mode
conformed: 19 packets, 9424 bytes; actions:
  transmit
exceeded: 0 packets, 0 bytes; actions:
  drop
violated: 0 packets, 0 bytes; actions:
  drop
color aware: No
green_qos_group: 0, yellow_qos_group: 0
```

***** Bc as modified for hardware assist

Note Although we might slightly modify rate and burst parameters to accommodate the hardware assist, the platform always aims to retain the rates and resulting accuracy within 1% of what you request.

Percent-based Policer

The Percentage-based Policing feature enables you to configure traffic policing based on a percentage of the bandwidth available on the interface. Hence, you can use the same policy-map for multiple interface types with differing amounts of bandwidth. Recalculating the bandwidth for each interface or configuring a different policy-map for each type of interface is unnecessary.

Note If the interface is a shaped-ATM permanent-virtual circuit (PVC), we calculate the total bandwidth as follows:

- For a variable bit rate (VBR) virtual circuit (VC), the sustained cell rate (SCR) is used.
- For an available bit rate (ABR) VC, the minimum cell rate (MCR) is used.

You can use percentage-based policers for both CIR and PIR, calculating either from a specified percentage of either the interface bandwidth or parent shaper (if one exists).

With percent-based policing, if you choose to specify burst parameters (Bc and Be), they must be in ms rather than bytes. Given the speed of the target interface, IOS converts the value to bytes in two steps:

1. Using the speed of a target interface, IOS converts percentage to bps CIR
2. With bps CIR and *burst in time*, burst is converted to bytes.

Let's configure the Bc to 10 ms (relative to the police rate) and the CIR to 10% of the available interface bandwidth:

```
policy-map police-percent
  class class-default
    police cir percent 10 bc 10 ms
```

If we apply police-percent to a GigabitEthernet interface (1Gbps nominal bandwidth), IOS converts the CIR to 100 Mbps and the Bc to 125,000 bytes (100 Mbps x 10msec / 8):

```
show policy-map interface GigabitEthernet1/0/0

  Service-policy input: police-percent
```

```
    Class-map: class-default (match-any)
      834 packets, 413664 bytes
      5 minute offered rate 13000 bps, drop rate 0000 bps
      Match: any
      police:
        cir 10 %, bc 10
        cir 100000000 bps, bc 125000 bytes  Configured CIR and Bc converted to bps and
bytes, respectively.
```

Now, if we attach police-percent to a POS OC3 interface, the rate will be based on a nominal bandwidth of 155 Mbps. CIR will be calculated as 15.5 Mbps; the Bc, 19375 bytes (15.5 Mbps x 10msec / 8):

```
show policy-map interface POS1/1/0

  Service-policy input: police-percent

    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0000 bps, drop rate 0000 bps
      Match: any
      police:
          cir 10 %, bc 10
          cir 15500000 bps, bc 19375 bytes  Configured CIR and Bc converted to bps and
bytes, respectively.
```

Color-Aware Policers

A *color-aware policer* accounts for any preexisting markings that were determined by a previous node's policer as *in-contract* or *out-of-contract* (the previous node is typically at the edge of the network). Where color-aware policer is configured, we use such markings to determine the appropriate policing action for the packet. Traffic that was designated out-of-contract will always remain out-of-contract. Traffic that was designated in-contract may be demoted to out-of-contract by the new policer.

The ASR 1000 provides a limited implementation of color-aware policing; we restrict the contents of the class-maps used to determine the existing color of traffic:

- Only QoS group matching is supported in color-aware class-maps (only classification based on qos-group is supported.)
- Only one filter (one **match qos-group** *value* statement) is supported per color-aware class.

  You can use a child policy to set the qos-group based on a field you want in the received packet.
- Color-aware "specific" statistics are not supported.
- You cannot use the **no class-map** command to remove a color-aware map provided it is referenced in a color-aware policer. You must first remove all color-aware policers (using either the **no conform-color** or the **no exceed-color** command).

The "color" in color aware policing refers to how we educate the policer on how to interpret pre-existing markings in a received packet. Typically, we use Green to represent traffic that was pre-marked as *conforming* or in-contract. Similarly, Yellow represents traffic that was pre-marked as *exceeding* or out-of-contract.

Note that Green or Yellow are representative only; the CLI uses *conform-color* and *exceed-color* instead. Through the **police** command, these keywords specify class-maps that are used to determine the pre-existing color of that packet.

The following example shows how a child policy-map enables you to specify pre-existing color based on any field in the received packet. The color-aware policer is configured in a class that matches all packets from one of the DSCP assured forwarding traffic classes, AF4.

For this example, a packet marked AF41 is in-contract (conform or green), AF42 is out-of-contract (exceed or yellow) and AF43 is violating. The child policy mark-existing-color classifies packets based on the received DSCP, internally marking AF41 packets as qos-group 1 and AF42 packets as qos-group 2.

The color-aware policer will use the pre-conform (classify green packets) and pre-exceed (classify yellow packets) class-maps to determine the existing color of an arriving packet. Although these class-maps only support the qos-group filter, use of the child policy allows us to determine the pre-existing color based on the DSCP value in the received packet:

```
class-map af4
  match dscp af41 af42 af43
!
class-map af41
  match dscp af41
class-map af42
  match dscp af42
!
class-map pre-conform                !These are policer
  match qos-group 1                  !class-maps that
class-map pre-exceed                 !only support qos-group
  match qos-group 2
!
policy-map mark-existing-color       !We use a child policy
  class af41                         !to set qos-group
    set qos-group 1                  !based on DSCP in the
  class af42                         !received packet
    set qos-group 2
!
policy-map dual-rate-color-aware
  class af4
    police cir 1m bc 5000 pir 2m be 5000
      conform-action set-dscp-transmit af41
      exceed-action set-dscp-transmit af42
      violate-action drop
      conform-color pre-conform exceed-color pre-exceed
    service-policy mark-existing-color
```

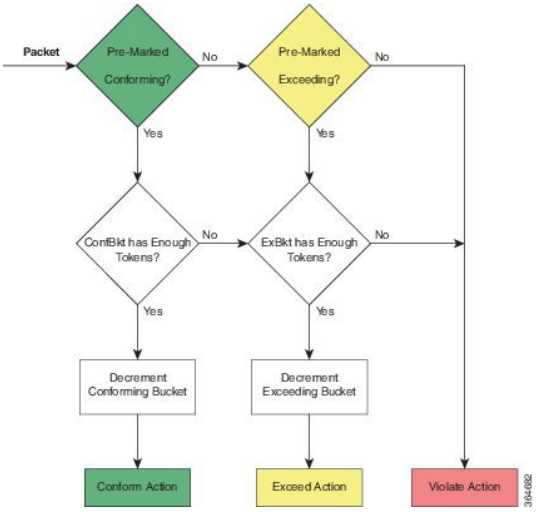
Single-Rate, Color-Aware, Three-Color Policer

The *color-aware mode* of a single-rate, three-color policer extends the standard single-rate, three-color policer. (See Single-Rate, Three-Color Policer, on page 264.)

Similar to the "color-blind" version of this type of policer, we maintain and replenish two distinct token buckets for the "color-aware" mode. The difference stems from how a packet is evaluated against these buckets. Recall that a color-aware policer honors any decision made by a previous router (the current designation of a packet) and ensures that the decision of a previous router is not undone (an exceeding or violating packet can never be promoted to conforming).

The following flowchart illustrates the algorithm used for handling traffic in single-rate, color-aware traffic policing. ConfBkt represents the conforming token bucket and ExBkt the exceeding token bucket.

Figure 68: Single-Rate, Color-Aware, Three-Color Policer



When a packet arrives, the policer uses its color-aware class-maps to determine the pre-existing color of that packet. This color may be conforming (matches the conform-color class-map), exceeding (matches the exceed-color class-map) or violating (matches neither of these class-maps).

If a packet is pre-marked as conforming it might end up as conforming, exceeding or violating. Evaluation proceeds as though the policer was operating in a color-blind mode.

- If the conforming token bucket has enough tokens the packet will take the conform action and the bucket will be decremented by the size of the packet.
- If the conforming token bucket has insufficient tokens but the exceeding token bucket does, the packet will take the exceed action and the exceeding token bucket is decremented by the size of the packet.
- If neither the conforming nor exceeding token bucket has sufficient tokens the packet will take the violate Action.

If a packet is pre-marked as exceeding it can never be promoted to conforming so evaluating the conforming token bucket is unnecessary.

- If the exceeding token bucket has sufficient tokens the packet will take the exceeding action and the bucket is decremented by the size of the packet.

If a packet is pre-marked as violating

- The violating action is taken and either token bucket is unchanged.

Dual-Rate, Color-Aware, Three-Color Policer

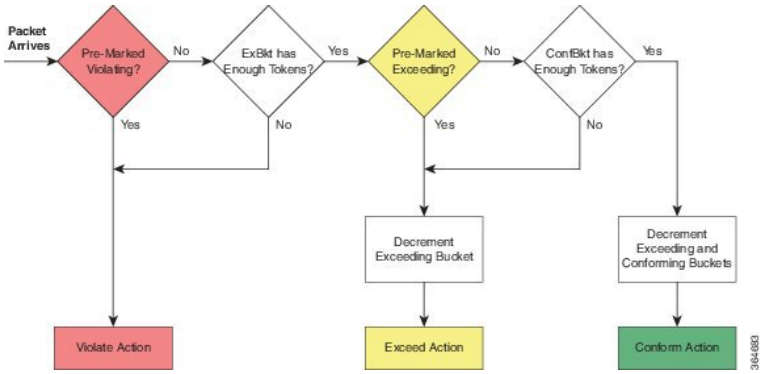
The color-aware mode of a dual-rate, three color policer extends the standard dual-rate, three-color policer. (Refer to Dual-Rate, Three-Color Policer, on page 266.)

Similar to the "color-blind" version of this type of policer, we maintain and replenish two distinct token buckets for the "color-aware" mode. The difference arises from how a packet is evaluated against these buckets. Recall that a color-aware policer honors any decision made by a previous router (the current designation of a packet)

and ensures that the decision of a previous router is not undone (an exceeding or violating packet can never be promoted to conforming).

The following diagram illustrates the algorithm used for handling traffic in dual-rate, color-aware policing. ConfBkt represents the conforming token bucket and ExBkt the exceeding token bucket.

Figure 69: Dual-Rate, Color-Aware, Three-Color Policer



When a packet arrives, the policer uses its color-aware class-maps to determine the pre-existing color of that packet. This color may be conforming (matches the conform-color class-map), exceeding (matches the exceed-color class-map) or violating (matches neither of these class-maps).

If a packet is pre-marked as violating

- we take the violating action and neither bucket is changed (decremented).

If a packet is pre-marked as exceeding

- and the exceeding bucket has sufficient tokens, the packet will remain as exceeding and we decrement the exceeding bucket by the size of the packet.
- and the exceeding bucket has insufficient tokens, the packet will take the violate action and neither bucket is changed.

If a packet is pre-marked as conforming

- and the exceeding bucket has insufficient tokens the packet will take the violate action and neither bucket is changed.
- and the exceeding bucket has sufficient tokens but the conforming bucket does not the packet will take the exceed action and we decrement the exceeding bucket by the size of the packet.
- and both the exceeding and conforming buckets have sufficient tokens the conform action will be taken and we decrement both buckets by the size of the packet.

Hierarchical Policy Containing Policers

In hierarchical traffic policing, we introduced hierarchical policies as a way to offer more granular control over traffic classes and to have some QoS actions operate on the aggregate of a number of those classes.

The ASR 1000 Series supports at most three levels in a hierarchical policy and the policing feature (one particular QoS action) can be configured at any level of that policy.

When describing hierarchical policies we often use different language to describe the distinct levels within that hierarchy (e.g., Top/Middle/Bottom, Parent/Child/Grandchild, Root/Leaf, Child/Parent/Grandparent). Because this can lead to ambiguity, we will always refer to the levels as Parent/Child/Grandchild where meanings are defined as follows:

Parent policy is a policy-map that will be attached to an interface using the **service-policy** command.
Child policy is a policy embedded directly in a class of the parent policy (using the **service-policy** command within a class).
Grandchild policy is a policy embedded directly in a class of the child policy.

Occasionally, we will refer to a policy's child or parent. They represent more relative terms (e.g., the parent of the *grandchild policy* references the child policy when we communicate in absolute terms).

Ingress Hierarchical Policy Containing only Policers

One of the simplest and perhaps most typical use of policers in hierarchical policies is an ingress policy containing only policers. We have already described how a policer is often used for network admission control (defined in Why Traffic Policing, on page 259). Replacing a simple policer with hierarchical policers allows the network operator to not only set an aggregate rate for network admission but also to specify rates for the individual classes of traffic that will be carried over the network.

For example, consider the following policy:

```
policy-map child
  class voice
    police cir percent 10 bc 5 ms
 !
policy-map parent
  class class-default
    police cir 50000000
    service-policy child
```

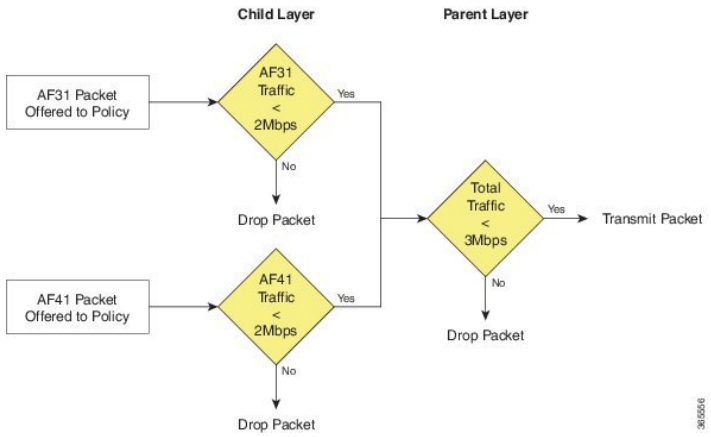
The policy-map parent, which is attached to the interface, defines the aggregate network admission rate (or service rate) for a customer so connected. In this example, the customer has contracted for 50 Mbps of network service. Within that network rate, the child policy limits individual classes of traffic. For example, the voice class specifies that traffic arriving at a rate exceeding 5 Mbps (10% of the parent) would simply be dropped.

Hierarchical Policers Order of Operation

On the ASR 1000 Series Appregation Services Router, we evaluate hierarchical policers for the child first then the parent. Although this scheme differs from IOS classic, it provides a much more meaningful construct. The following example should clarify this notion:

```
policy-map child
  class AF41
    police 2m
  class AF31
    police 2m
!
policy-map parent
  class AF41_or_AF31
    police 3m
    service-policy child
```

Figure 70: Hierarchical Policing



If a packet arrives with a marking of AF31 it must first pass through the AF31 policer in the child policy, which allows such packets to pass through up to a rate of 2 Mbps. The packet must then pass through the parent policer, which observes both AF31 and AF41 traffic.

The combined rate of AF31 and AF41 traffic from the child policy could be up to 4 Mbps as each has a 2 Mbps policer configured. Although a packet passed through the child policer, it may be dropped by the parent policer if the rate arriving at that policer is above the configured 3 Mbps rate.

When policers are used in an egress policy-map with scheduling semantics (bandwidth/shape/priority) all policers will be evaluated before a packet is enqueued. Furthermore, a policer in the parent level would be enforced before a shape value in the child level (scheduling happens).

Percent-Based Policer in Hierarchical Polices

If a percent-based policer is used in the parent level of a policy-map the meaning of the percent is fairly intuitive - it is a percent of the bandwidth available in the interface where the policy-map is attached. When we use a percent-based policer in the child or grandchild level, the meaning can be a bit more ambiguous.

If the percent-based policer is configured in the child level it examines the parent level class to assess whether the bandwidth of that class has been constrained by either a shaper or policer. If so, the child policer CIR is a percent of the shape or police rate configured in the parent level. If not, the percent is interpreted as percent of the bandwidth available in the interface where the policy-map is attached.

If the percent-based policer is configured in the grandchild level it first looks at the child level class for a shaper or policer. If it finds one, it uses that rate in the child level. If none exists, the grandchild policer looks at its class in the parent level. It either finds a rate there or uses the rate of the interface to which the policy is attached.

If the percent policer is configured in the grandchild level and a rate-limiting feature (e.g., shaper or policer) is configured in both the child and parent levels, the grandchild always uses the rate configured in the child level. This is crucial as the sum of shapers or policers at any level in a policy can be greater than the physical bandwidth available.

If both a shaper and a policer are configured in the parent of a class with a percent-based policer, the percent-based policer is based on the lower rate configured (shaper or policer).

The following hierarchical policy-map illustrates these considerations:

```
policy-map grandchild
  class AF11
    police cir percent 60
```

```
  class AF12
    police cir percent 40
!
policy-map child
  class AF1
    bandwidth percent 50
    service-policy grandchild
!
policy-map parent
  class class-default
    shape average 50000000
    service-policy child
```

The policers in the grandchild policy are percent-based policers. They defer to their parent class (class AF1 in the parent policy) for rate-limiting features. Because none exist here, the policers "step up" to the parent class (**class class-default**, in the parent policy).

There, they find a shaper that limits the throughput to 50Mbps. So, the policer in class AF11 would be configured with a CIR of 30 Mbps (60% of 50Mbps); the policer in class AF12, a CIR of 20 Mbps.

Verifying the Configuration and Operation of the Policing Feature

As with all MQC QoS features, you have three ways to verify the configuration and performance of the policing feature:

- **show policy-map** *policy-name*

  Displays the user-entered configuration. Analogous to contents of the running configuration on the router but displays default values and actions not explicitly called out in the configuration.

- **show policy-map interface** *interface-name*

  Displays statistics for all features within that policy-map. Primary means of verifying that a QoS policy is operating as expected..

- **show platform hardware qfp active feature qos interface** *interface-name*

  Displays real-time information from the dataplane. Shows the exact rates and burst sizes that are programmed in hardware.

Example 1: show policy-map *policy-name* Command

If we configure the policy-map simple_policer as follows:

```
policy-map simple_policer
  class AF1
    police cir 20000000
```

show policy-map command output looks like this:

```
show policy-map simple_policer

Policy Map simple_policer
  Class AF1
```

```
    police cir 20000000 bc 625000
      conform-action transmit
      exceed-action drop
```

Besides the explicit conforming burst and conform (or exceed) actions, notice the lack of statistics or interface information. We merely define a policy, an action applicable to multiple interfaces.

Example 2: show policy-map interface *interface-name* Command

Here is a sample output of the **show policy-map interface** command for an instance of a policy-map attached to a particular interface:

```
show policy-map interface GigabitEthernet1/0/0

GigabitEthernet1/0/0

 Service-policy input: simpler_policer

  Class-map: AF1 (match-any)                              --+
   1000 packets, 1496000 bytes                              |Classification
   5 minute offered rate 0000bps, drop rate 0000bps         |Section
   Match: :dscp af11 (10) af12 (12) af13 (14)               |
   police:                                                 --+
     cir 20000000 bps, bc 625000 bytes                      |
    conformed 447 packets, 668712 bytes; actions:           |Policing
     transmit                                               |Section
    exceeded 553 packets, 827288 bytes; actions:            |
     drop                                                   |
    conformed 0000 bps, exceeded 0000 bps                  --+

  Class-map: class-default (match-any)
    0 packets, 0 bytes
    5 minute offered rate 0000 bps, drop rate 0000 bps
    Match: any
```

The organization of the output reflects the policy-map definition combined with a hierarchical output that represents the policy-map and class hierarchy. Within each class, a classification section displays the classification counters (statistics of packets that were determined to belong to this class) and the classification criteria (a summary of the class-map that defines what packets belong to this class). Following the classification section, you observe a block that represents each QoS action configured within that class. Because policing is the sole action in this example, only a block of policing statistics displays.

The following table summarizes the meaning of different fields in the **show** command output.

SUMMARY STEPS

1. As you will observe, the output provides a summary of the configuration along with statistics. A router uses the statistics (over time) to calculate rates and display them. Rates in these formulations represent a *decayed average (rate)*. The frequency (default, 300 seconds) of the calculation hinges on the load-interval for that interface. Statistics in the **show policy-map interface** output persist until you issue a **clear counters** command. The dataplane updates the statistics every 10 seconds.
2. Notice that Classification and Policer Action statistics arise from different entities in the dataplane. Consequently, they might update at slightly different times (briefly, the action counters might exceed the classification counters).

DETAILED STEPS

	Command or Action	Purpose
Step 1	As you will observe, the output provides a summary of the configuration along with statistics. A router uses the statistics (over time) to calculate rates and display them. Rates in these formulations represent a *decayed average (rate)*. The frequency (default, 300 seconds) of the calculation hinges on the load-interval for that interface. Statistics in the **show policy-map interface** output persist until you issue a **clear counters** command. The dataplane updates the statistics every 10 seconds.	
Step 2	Notice that Classification and Policer Action statistics arise from different entities in the dataplane. Consequently, they might update at slightly different times (briefly, the action counters might exceed the classification counters).	

Example 3: show platform hardware qfp active feature qos interface Command

This command should only be necessary if you believe the router is configured correctly but is behaving unexpectedly. Viewing information directly from the dataplane can help you assess whether any quantization of rates or burst parameters were necessary to accommodate the hardware.

The following example corresponds to the **show policy-map interface** output from the previous example:

```
show platform hardware qfp active feature qos interface g1/0/0

Interface: GigabitEthernet1/0/0, QFP interface: 9
  Direction: Input
    Hierarchy level: 0
    Policy name: simple_policer
      Class name: AF1, Policy name: simple_policer
        Police:
            cir:  20000000 bps,  bc: 638976 bytes
            pir: 0 bps,  be: 638976 bytes
            rate mode: Single Rate Mode
            conformed: 447 packets, 668712 bytes; actions:
              transmit
            exceeded: 427 packets, 638792 bytes; actions:
              drop
            violated: 126 packets, 188496 bytes; actions:
              drop
            color aware: No
            green_qos_group: 0, yellow_qos_group: 0
      Class name: class-default, Policy name: simple_policer
```

If you understand the output of the previous two commands (Example 1 and Example 2), this output should be pretty self-explanatory. However, you should be aware of the following points related to using this command:

Although we configured a single-rate two-color policer, the output of the dataplane command corresponds to a single-rate, three-color policer. The hardware always operates in a three-color mode. To achieve two-color functionality it simply matches the Violate and Exceed actions. When we push statistics to the control plane, IOS aggregates the Exceed and Violate statistics to generate the expected appearance of a two-color policer.

Statistics in the dataplane are transitory. Every 10 seconds the dataplane pushes statistics to IOS and then clears its local counters. Essentially, all statistics observed through the dataplane command are counts of what transpired since the last push. This means that dataplane commands help you view hardware behavior in real time. For meaningful (persistent) statistics, however, you should always use the regular IOS **show policy-map interface** command.

Configuration Examples for QoS Packet Policing

Example 1: Simple Network Admission Control

In its simplest form a policer can be used to rate-limit all traffic entering an interface (and thereby a network). We assume that the network sending the traffic will "shape" what exits its egress interface and only send traffic that will conform to the contracted rate. We can use *egress scheduling* on the senders' network to apportion the contracted rate to different classes of traffic.

With this simplest example of policing no classification is required as the policer is intended to cap all traffic. We will consider all traffic as belonging to class-default in the absence of any user-defined classes.

In the following example, we have a GigabitEthernet connection but the customer has only contracted for a 100 Mbps service rate. The configuration could look something like this:

```
policy-map ingress_cap_all_100m
  class class-default
    police cir 100000000
!
interface GigabitEthernet1/0/0
  service-policy ingress_cap_all_100m
```

Example 2: Network Admission Control - Hierarchical Policers

In Example 1: Simple Network Admission Control, on page 283 we policed all traffic to a contracted service-rate, assuming that the sender would apportion bandwidth within that contracted rate. However, we may not always trust the sender to limit the traffic within an individual class. For example, say we offer a priority service (traffic guaranteed low latency through the network) but charge the user for different levels of priority access. By simply applying the simple policer in Example 1 we could not guarantee that the sender doesn't forward us more priority traffic than contracted. We can expand the example to enforce also a cap on an individual class of traffic.

In the following example, we limit the total admission to 100 Mbps AND ensure that voice traffic caps at 5 Mbps of traffic:

```
class-map match-all voice
  match dscp ef
!
! child policy to enforce 5Mbps Voice Traffic
!
policy-map ingress_police_child
  class voice
    police cir percent 5 bc 5 ms
!
policy-map police_ingress_parent
  class class-default
    police cir 100000000
  service-policy ingress_police_child
```

```
!
interface GigabitEthernet1/0/0
  service-policy in police_ingress_parent
```

Example 3: Network Admission Control - Color-Aware Policer

In Example 2: Network Admission Control - Hierarchical Policers, on page 283, we introduced the scheme of capping a particular class of traffic within the contracted service-rate. This scheme hinges on customer shaping of traffic to the service-rate.

If we received traffic at a rate exceeding the parent policer (the contracted service-rate), no guarantee exists that it would not drop some of the voice traffic admitted by the child policer. To ensure that any traffic admitted by the child policer is also admitted by the parent, you could employ a color-aware policer for the parent policer.

The following example shows how complex outcomes can be achieved with combinations of policers. Here, we mark all voice traffic admitted by the child policer as Green (qos-group1) and all traffic other than voice as Yellow (qos-group2). The parent policer is configured with a CIR that ensures that we forward all the Green traffic and a PIR that ensures that we enforce the contracted service-rate:

```
class-map match-all voice
  match dscp ef
!
!child policy to enforce 5Mbps Voice Traffic
!
policy-map ingress_police_child
  class voice
    !conforming voice marked Green, Excess Dropped
    police cir 5m bc 3125 conform-action set-qos-transmit 1
  class class-default
   !all traffic other than voice marked Yellow
   set qos-group2
!
class maps needed for color-aware policer
!
class-map policer-green
  match qos-group1
class-map policer-yellow
  match qos-group2
!
!parent policy to enforce 100Mbps service rate
!
policy-map ingress_police_parent
  class class-default
    police cir 5m bc 3125 pir 100m be 625000
      conform-action transmit
      exceed-action transmit
      violate-action drop
      conform-color policer-green exceed-color policer-yellow
    service-policy ingress_police_child
!
interface GigabitEthernet1/0/0
  service-policy in ingress_police_parent
```

Command Reference

police

As discussed within chapter there are three variants of the police command that achieve the same result, namely:

[no] police *cir* [**bc** *conform-burst*] [**pir** *pir*] [**be** *peak-burst*] [**conform-action** *action* [**exceed-action** *action* [**violate-action** *action*]]]

[no police cir *cir* [**bc** *conform-burst*] [**pir** *pir*] [**be** *peak-burst*] [**conform-action** *action* [**exceed-action** *action* [**violate-action** *action*]]]

[no] police rate*cir* [**bc** *conform-burst*] [**pir** *pir*] [**be** *peak-burst*] [**conform-action** *action* [**exceed-action** *action* [**violate-action** *action*]]]

Henceforth we shall denote this as:

[no] police [cir | rate]*cir* [**bc** *conform-burst*] [**pir** *pir*] [**be** *peak-burst*] [**conform-action** *action* [**exceed-action** *action* [**violate-action** *action*]]]

We have already seen how the same command may be used to configure different types of policers.

Rather than present a single CLI which would be confusing and option combinations which might not be correct, we will present a subset of the options depending on the policer type you wish to configure.

Single-Rate, Two-Color Policer

This policer type can be expressed on a single line if you require only one action per designated conformance level:

[no] police [cir | rate]*cir***[percent** *percent***][[bc]** *conform-burst* **[ms]**] [**account** *options*]**[conform-action** *action* **[exceed-action** *action***]]**

Multiple lines (using sub-modes) are necessary if we requiremore than one action:

[no] police [cir | rate]*cir***[percent** *percent***][[bc]** *conform-burst* **[ms]**][**account** *options*] **<return>**
[no] conform-action *action* **<return>**
[no] conform-action *action* **<return>**
[no] exceed-action *action* **<return>**
[no] exceed-action *action* **<return>**

Single-Rate, Three-Color Policer

This policer type can be expressed on a single line if you require only one action per designated conformance level:

[no] police [cir | rate]*cir***[percent** *percent***][[bc]** *conform-burst* **[ms]**] [**[be]** *exceed-burst* **[ms]**][**account** *options*]**conform-action** *action* **exceed-action** *action* **[violate-action** *action***]**

Multiple lines (using sub-modes) are necessary if we require more than one action:

[no] police [cir | rate]*cir***[percent** *percent***][[bc]** *conform-burst* **[ms]**] [**[be]** *exceed-burst* **[ms]**][**account** *options*] **<return>**
[no] conform-action *action* **<return>**
[no] conform-action *action* **<return>**
[no] exceed-action *action* **<return>**
[no] exceed-action *action* **<return>**
[no] violate-action *action* **<return>**
[no] violate-action *action* **<return>**

Dual-Rate, Three Color Policer

This policer type can be expressed on a single line if you require only one action per designated conformance level:

[no] police [cir | rate]*cir***[percent** *percent***][[bc]** *conform-burst* **[ms]**] **[pir]** *peak-rate* **[ms][[be]** *exceed-burst* **[ms]**][**account** *options*]**conform-action** *action* **exceed-action** *action* **[violate-action** *action***]**

Multiple lines (using sub-modes) are necessary if we require more than one action:

[no] police [cir | rate]*cir***[percent** *percent***][[bc]** *conform-burst* **[ms]**] [**[pir]** *peak-rate* **[ms]**][**[be]** *exceed-burst* **[ms]**][**account** *options*] **<return>**
[no] conform-action *action* **<return>**
[no] conform-action *action* **<return>**
[no] exceed-action *action* **<return>**
[no] exceed-action *action* **<return>**
[no] violate-action *action* **<return>**
[no] violate-action *action* **<return>**

Single-Rate, Three-Color, Color-Aware Policer

[no] police [cir | rate]*cir***[percent** *percent***][[bc]** *conform-burst* **[ms]**] [**[be]** *exceed-burst* **[ms]**][**account** *options*] **<return>**
[no] conform-action *action* **<return>**
[no] conform-action *action* **<return>**
[no] exceed-action *action* **<return>**
[no] exceed-action *action* **<return>**
[no] violate-action *action* **<return>**
[no] conform-color *conform-color* **exceed-color** *exceed-color***<return>**

Dual-Rate, Three-Color, Color-Aware Policer

[no] police [cir | rate]*cir***[percent** *percent***][[bc]** *conform-burst* **[ms]**][**[pir]** *peak-rate* **[ms]**] [**[be]** *exceed-burst* **[ms]**][**account** *options*] **<return>**
[no] conform-action *action* **<return>**
[no] conform-action *action* **<return>**
[no] exceed-action *action* **<return>**
[no] exceed-action *action* **<return>**
[no] violate-action *action* **<return>**
[no] conform-color *conform-color* **exceed-color** *exceed-color***<return>**

police Command Default and Modes; Keyword/Argument Descriptions

Command Default

Disabled

Command Modes

Policy-map class configuration (config-pmap-c) when specifying a single action to be applied to a marked packet

Policy-map class police configuration (config-pmap-c-police) submode when specifying multiple actions to be applied to a marked packet

Syntax Description

The following table list the keywords/arguments for the **police** command and their purpose.

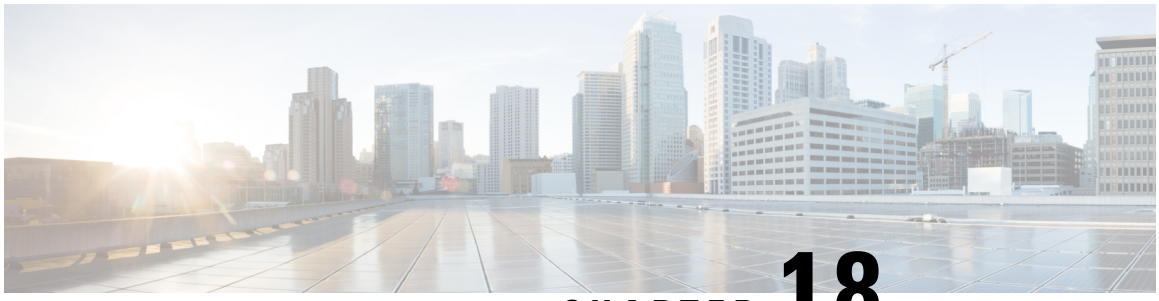
Keyword/Argument	Definition
bc	Specifies the Conforming Burst Size(Bc).
be	Specifies the Exceeding Burst Size(Be).
cir	Specifies the Committed Information Rate(CIR).
Conform-Action	Specifies the action to take on traffic that is determined to be "conforming."
Exceed-Action	Specifies the action to take on traffic that is determined to be "exceeding."
pir	Specifies the Peak Information Rate (PIR).
Violate-Action	Specifies the action to take on traffic that is determined to be "violating."

The following table lists the options for the Account keyword.

Table 17: Account keyword options

Option	Purpose
qinq	Specifies queue-in-queue encapsulation as the BRAS-DSLAM encapsulation type
dot1q	Specifies IEEE 802.1Q VLAN encapsulation as the BRAS-DSLAM encapsulation type
aal5	Specifies the ATM Adaptation Layer 5 that supports connection-oriented variable bit rate (VBR) services
aal3	Specifies the ATM Adaptation Layer 3 that supports both connectionless and connection-oriented links

Option	Purpose
subscriber-encapsulation	Specifies the encapsulation type at the subscriber line
user-defined	Indicates that the router is to use the offset value that you specify when adjusting policing length
offset	Specifies the number of bytes that the router is to use when calculating overhead. Valid values are from -63 to 63 bytes
atm	Applies the ATM cell tax in the ATM overhead calculation



CHAPTER 18

Queue Limits and WRED

About

On Cisco IOS XE devices, we dedicate memory to store packets that are queued in egress interface or QoS queues. The memory is treated as a global pool available to all interfaces rather than as carved or owned by individual interfaces.

A *queue-limit* caps the depth of a particular queue and serves two purposes. First, they constrain how much of the available packet memory an individual queue may consume. This ensures that other interfaces or queues also have fair access to this shared resource. Second, they constrain how much data we store if a queue is congested, thereby capping the latency applications in that queue will experience.

When a packet is ready for enqueuing we check the current depth of that queue and the configured queue-limit. If the former has already achieved the latter then the packet is dropped (tail drop).

Queue Limits

The packet memory of an ASR 1000 Series Aggregation Services Router (heretofore the ASR 1000 Series Router) is a shared resource. We do not allocate individual interfaces and queues a share of this memory. Rather they represent a global-pool available to all queues on a first come, first serve basis.

To control how much data an individual queue may store in the shared packet memory, we use *queue-limit*, a per-queue configurable value. It serves two purposes. First, it limits the latency for a packet arriving to a nearly full queue - at some point it is better to drop than to deliver packets so slowly that they are useless at the receiver. Second, it ensures that a single interface can't put so many packets into the shared memory that it starves other interfaces.

We manage the shared memory very efficiently: Instead of carving pools of buffers into predetermined sizes, the hardware manages blocks of memory (32 byte blocks on the original QFP) and assigns the minimum number of blocks needed to store a packet.

The following table shows how the amount of packet memory and the maximum configurable number of queues vary by platform:

ESP (Embedded Services Processors) Router Hardware	Packet Memory	Maximum Queues
ASR1001	64 MB	16,000
ASR1001-X	512 MB	16,000
ASR1002-F	64 MB	64,000
ASR1002-X	512 MB	116,000
ESP5	64 MB	64,000
ESP10	128 MB	128,000
ESP20	256 MB	128,000
ESP40	256 MB	128,000
ESP100	1 GB (two 512-MB)	232,000*
ESP200	2 GB (four 512-MB)	464,000*

For ESP100 and ESP200, physical ports are associated with a particular QFP (Quantum Flow Processor) complex on the ESP card. To maximally-use all queues, you must distributed them among different slots and SPAs in the chassis.

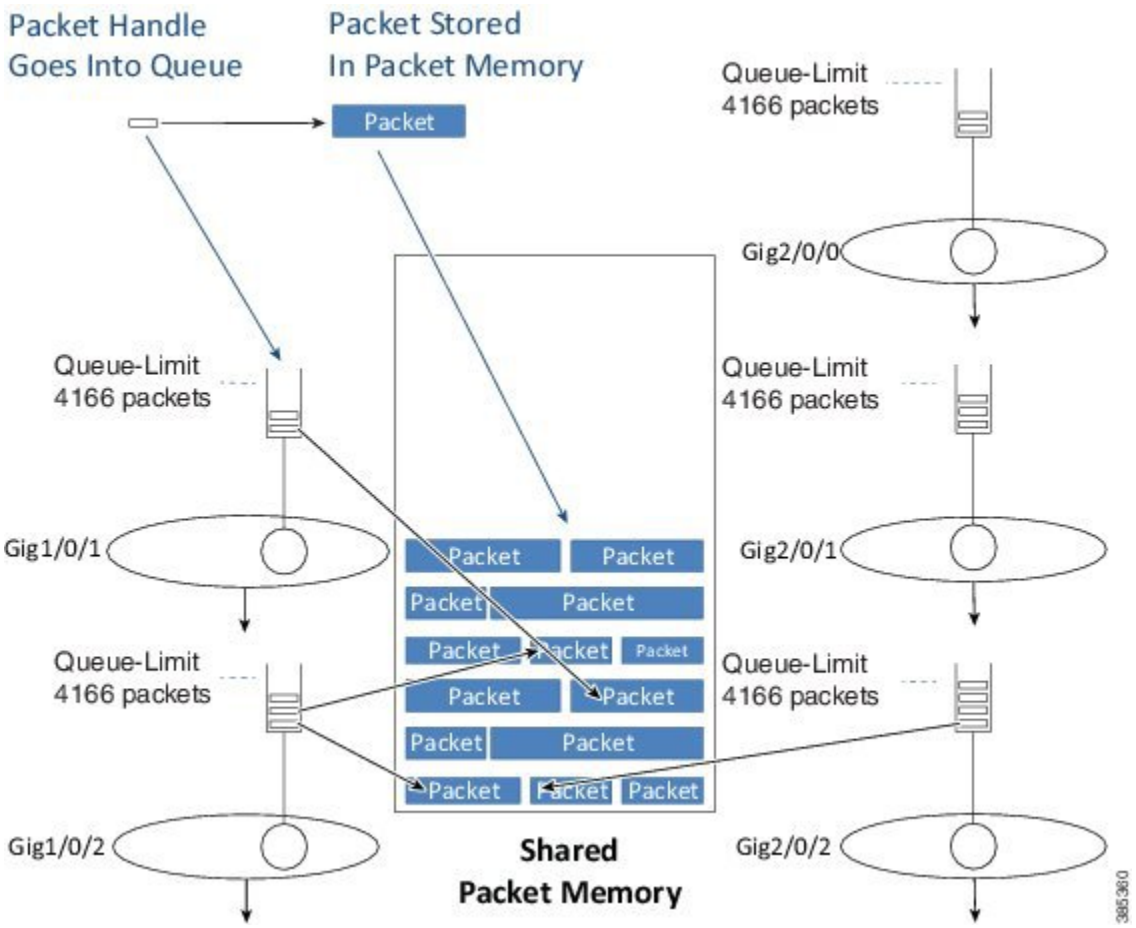
The amount of packet memory in an ASR 1000 Series Router is driven by a number of factors: cost, technology availability and what makes sense. When QFP was first released few choices for memory technologies were available to handle the 10s of gigabits per second of reads (and writes) required for packet memory. Even when memory could handle the speed requirements, options for size were limited and module cost was extremely high; we could have designed a system with more memory but it would have been prohibitively expensive with no real upside.

Beyond simply the number of queues supported, you must also consider the rate at which packets can ingress and egress the system. For example, looking at the ESP10, you could say that 128MB and 128,000 queues translate into 1KB of memory per queue. This is pretty meaningless if you never have all 128K queues congested simultaneously.

Let's view the size in another way: An ESP10 can transmit or receive data at a max rate of 10Gbps. At this speed, 128 MB of memory provides over 100mS of buffering which is quite reasonable.

From above it should now be evident that we expect to oversubscribe the sum of all queue-limits in the system.

Figure 71: Queue limits

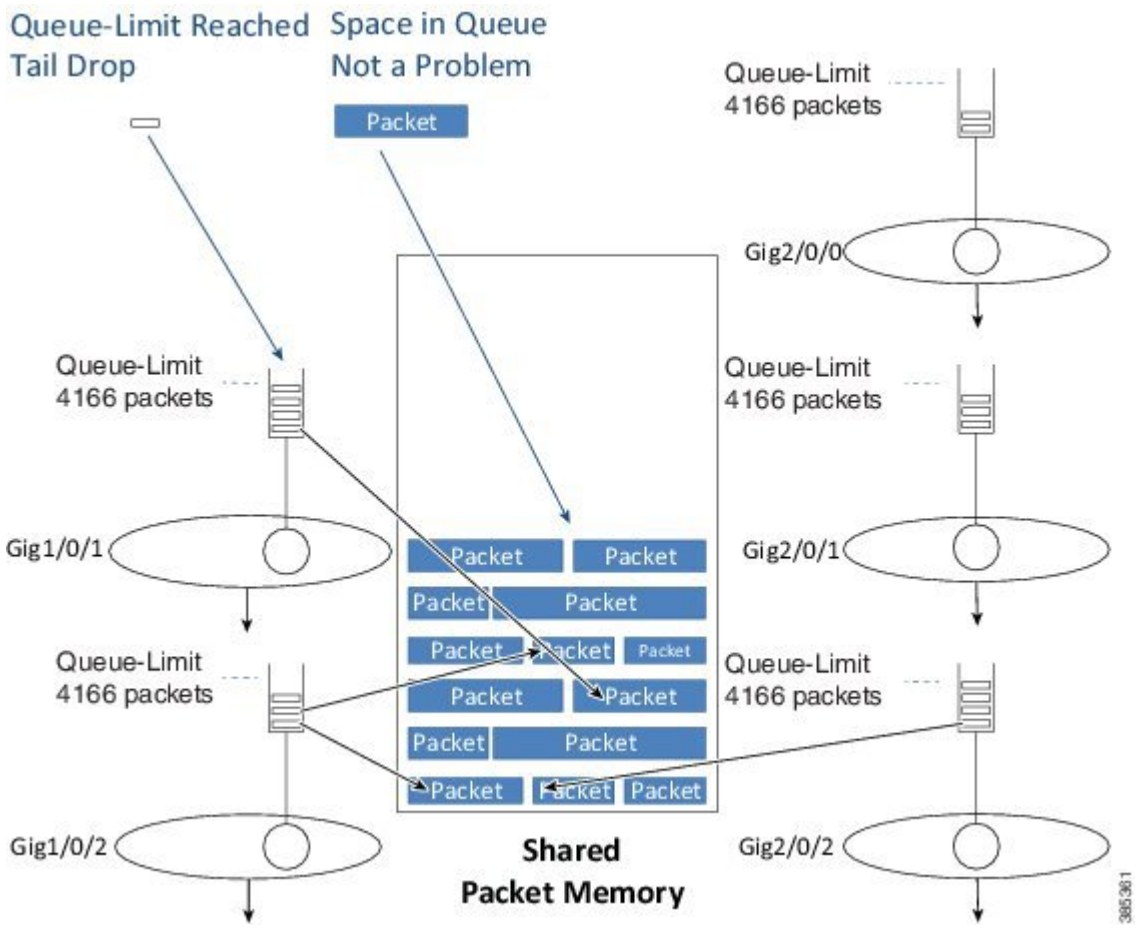


After we determine the egress interface for a packet, we will know the queue information for that interface. Into that queue we place a small packet handle, which contains the scheduling length for the packet, and a pointer to where the packet is stored in the shared packet memory. We store the actual packet itself in shared packet memory.

Tail Drop

When we enqueue a packet we first examine the configured queue-limit as well as how much data that interface currently has buffered (the *instantaneous queue depth*).

Figure 72: Tail Drop



If the queue depth is already at the preconfigured limit, we will drop the packet and record a **tail drop**.

If QoS is not configured, you can view the drop in the output of the **show interface** command.

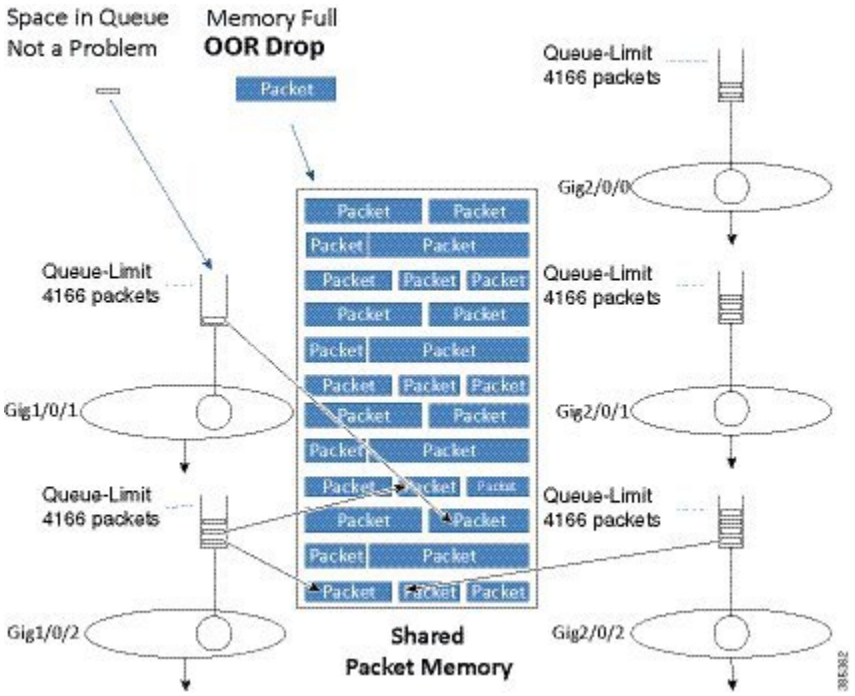
If QoS is configured, you can view the drop in the class output of the **show policy-map interface**.

As the diagram illustrates, a tail drop does not mean that no memory exists to store the packet rather it means that a queue has already reached its individual limit on how much data it can store.

Out of Resources Drop

Another scenario is possible on enqueue if the queue has not yet reached its individual queue-limit but the shared-packet memory may be full. If so and no place exists to store the packet, we must drop it. This drop would be recorded as a No Buffer drop and reported to the syslog as an *Out Of Resources (OOR) condition*.

Figure 73: ORR Drop

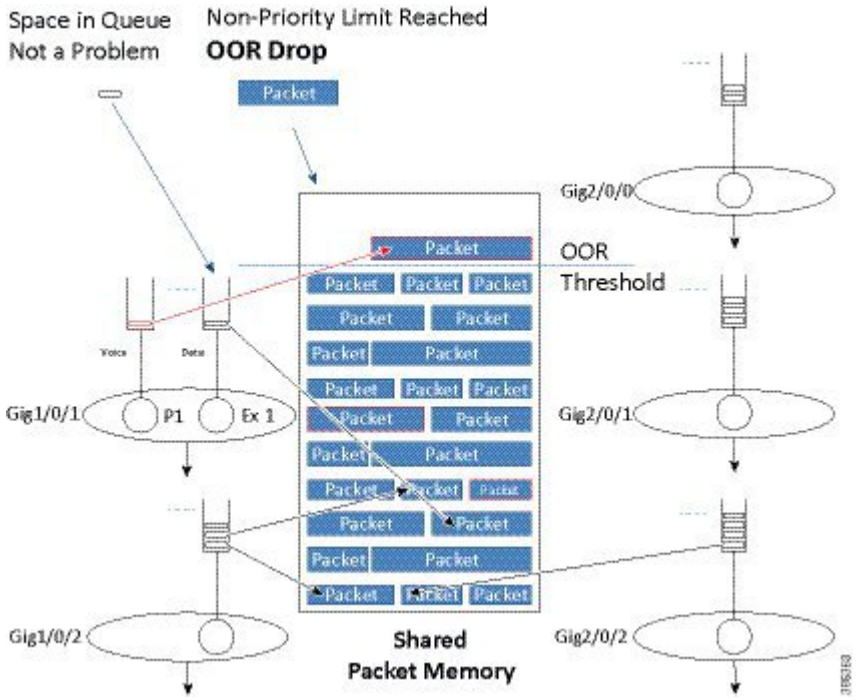


If OOR drops are only seen very occasionally you can ignore them. However, if this is a regular condition then you should review queue-limits to see whether you are allowing an individual queue or interface to consume too much memory. To avoid this situation, you might need to lower the queue-limit for one or more queues.

Memory Reserved for Priority Packets

The description of packet memory being 100% full is not really accurate. We know that some packets (those from priority classes and pak_priority packets) are more important than others and we want to ensure that we always have space in memory to store these important packets so. To do this, we limit packets from normal data queues to 85% of the total packet memory.

Figure 74: Memory Reserved for Priority Packets



The diagram above shows how we treat priority packets and data packets differently. In this scenario, 85% of the packet memory has been consumed. If a normal data packet arrives it is dropped because the OOR threshold has been reached. However, if a priority packet were to arrive it would still be enqueued as there is physical space available.

Please note that we are not restricting priority packets to a small portion of the memory. Instead, we are dropping non-priority packets when memory is nearly full.

Vital Threshold

We also provide a second level of protection that will drop all user traffic, including priority packets, when the memory utilization exceeds 98%. We term this the *vital threshold* and it ensures that we can enqueue internal control packets, which are inband packets that may need to travel between different control processors in the system. As priority packets are usually forwarded when enqueued, exceeding a 98% threshold is unexpected.

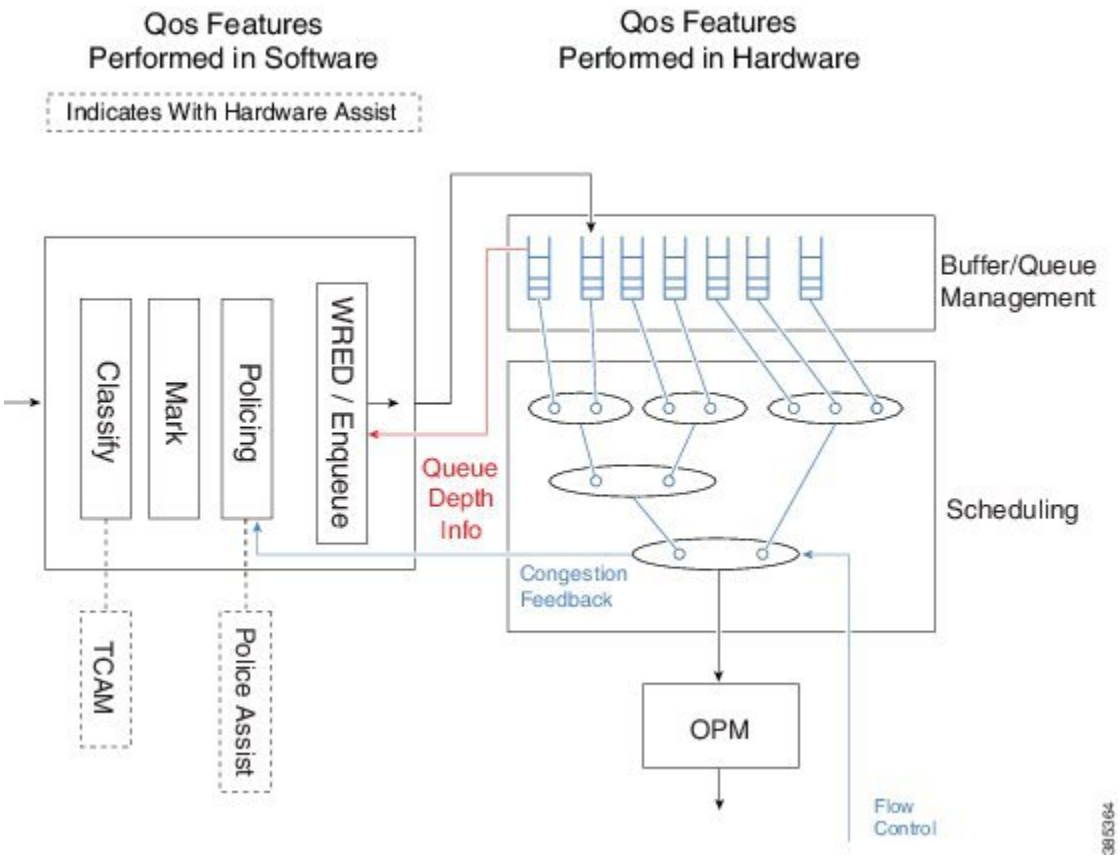
You can see the amount of memory in a system and the realtime-utilization of that memory using the **show platform hardware qfp active bqs 0 packet-buffer utilization** command.

```
show platform hardware qfp active bqs 0 packet-buffer utilization
Packet buffer memory utilization details:
  Total:     256.00 MB
  Used :    2003.00 KB
  Free :     254.04 MB

  Utilization:    0 %

  Threshold Values:
    Out of Memory (OOM)      :     255.96 MB, Status: False
    Vital (> 98%)            :     253.44 MB, Status: False
```

```
Out of Resource (OOR)    :     217.60 MB, Status: False
```

On the ASR 1000 Series Aggregation Services Router, all queuing, scheduling and packet memory management is performed by dedicated hardware. When we enqueue a packet we are passing control from software to hardware. As the hardware, specifically the BQS (Buffering, Queuing and Scheduling) subsystem, manages the memory it monitors how much data each queue is currently storing in packet memory. When we are ready to enqueue a packet we query the hardware for current status. The hardware will report an instantaneous and an average queue depth for that queue. Software will then determine whether to continue with the queue or drop the packet (and report it). Tail drop decisions are made using the instantaneous queue depth reported by hardware. Instead, WRED uses the average queue depth (see Average Queue Depth, on page 303).

Figure 75: Vital Threshold



Packet Mode vs Byte Mode

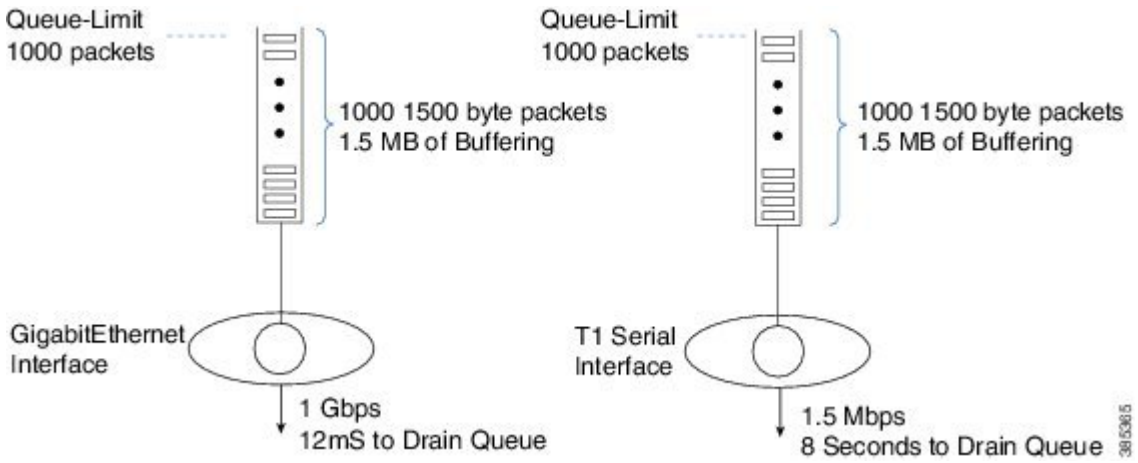
The hardware may operate in one of two modes; packet mode or byte mode. When reporting the instantaneous and average queue depth it will report those values in packets or in bytes but not in both. At the time a queue is created the mode is set and can't be changed unless you remove and reattach the policy-map.

The diagram above shows how some QoS features are performed in software and others in hardware. The enqueue is really on the boundary of the two. Software will receive Queue Depth Information from the hardware and then decide whether to drop the packet or to move it to packet memory and add a packet handle to the queue. WRED is a more advanced form of drop decision and will be covered later in the chapter.

Default Queue-Limits

The following diagram shows the need for *variable queue limits*.

Figure 76: Variable Queue Limits



The queue on the left is serviced at 1 Gbps. If 1000 1,500 byte packets were waiting transmission it would take 12 mS to drain the queue. This means a packet arriving to an almost full queue could be delayed by 12 mS while waiting its turn to be forwarded.

The schedule on the right represents a T1 interface, a considerably slower interface operating at approx. 1.5 Mbps. If the same 1000 packets were waiting transmission through a T1 interface it would take 8 seconds to drain the queue. Obviously most users (and applications) would be disappointed with such a delay.

The diagram highlights the second role of queue-limits mentioned above - constraining the latency for applications in a queue.

How we determine the default queue mode and queue-limit will vary depending on whether or not QoS is configured.

Note that we select default queue-limits to be appropriate for as many users as possible but that does not mean they are always the best choice. We cannot know how many physical and logical interfaces will be configured in a system, how bursty traffic will be in any queue, the latency requirements of applications in a queue, etc. The defaults are a good starting point but it is not unusual to tune queue-limits further.

When Qos is not Configured

In the scheduling chapter we have seen that when no QoS is configured all packets go through a single FIFO that we refer to as the Interface Default Queue. The queue-limit for the interface default queue is configured in bytes and is calculated as 50mS worth of buffering based on the interface speed (ESP-40 is an exception where 25mS is used).

As an example consider a GigabitEthernet interface. The interface speed is 1 Gbps but with internal overdrive we send at 1.05 Gbps:

50mS worth of buffering in bytes would be: 1.05 Gbps / 8 bits per byte * .05 seconds = 6,562,500 bytes

You can use the **show platform hardware qfp active infrastructure bqs queue output default interface gig1/0/0 | inc qlimit** command to view the queue-limit for an interface default queue.

When QoS is Configured

Note The default mode for any queue created using the MQC CLI is packet. (This is an historical artifact rather than an admission that packet mode is superior.)

Calculating queue-limit depends on a number of factors:

If the queue is a *priority queue* the default queue-limit is 512 packets. Yes. This is a large limit but we assume that these values are meaningless. Because queue admission control ensures that packets are enqueued at a rate lower than they will be transmitted, a priority queue should always be nearly empty. Thus, we can set the queue-limit arbitrarily large and use it across all interface speeds.

For a *bandwidth queue* we target a maximum of 50mS worth of data buffered but make an exception for low speed queues where this might represent a very small amount of data. To calculate how much data would be transmitted (in 50mS) we need to know the service speed. For an interface default queue (recall, the only game in town for scenarios without QoS) this is simple - a single queue 'owns' the entire bandwidth of the interface. When QoS is configured, the picture gets murky.

First, we need to introduce the concept of *visible bandwidth* , a value ascertained from the configuration that captures the service rate of a queue without accounting for the offered load. The table below shows how the visible bandwidth depends on the commands used:

Table 18: Representation of Visible Bandwidth Depends on the Commands used

Commands	Visible Bandwidth
shape	shape rate
bandwidth	bandwidth rate
shape and bandwidth	bandwidth rate
bandwidth remaining	Inherited directly from the parent. • If the policy-map is attached to a physical interface the value inherited would be the interface speed. • If the policy is a child policy with a parent shaper the visible bandwidth would be the parent shape rate.

Second, we need the Maximum Transmission Unit (MTU) for the interface where the policy is attached. As we are configuring a queue-limit in packets (recall that this is the default) and want to limit the potential latency, we look at a worst case scenario where a queue is full of MTU-size packets (view the MTU in the output of the **show interface** command).

Given the visible bandwidth, the MTU, and a maximum of 50mS worth of buffered data, we can calculate a queue-limit as follows:

queue-limit = (visible bandwidth / 8 bits) * 50mS / MTU

Let's consider a queue shaped to 100 Mbps on a GigabitEthernet Interface. The visible bandwidth would be the shape rate (100 Mbps) and the MTU would be 1500 bytes (what you expect on an Ethernet type interface):

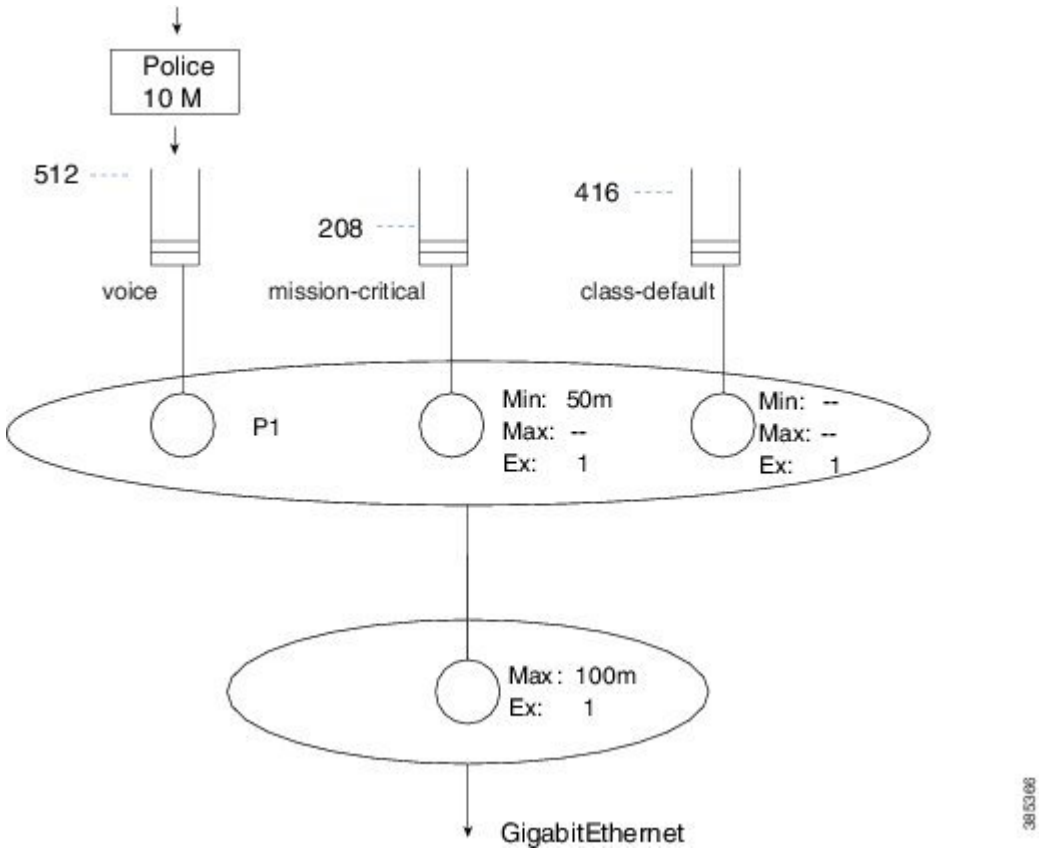
queue-limit = 100 Mbps / 8 bits * .05 sec / 1500 bytes = 416 packets

As mentioned, we make an exception for low speed queues. If the calculated queue-limit is less than 64 packets we use 64 packets as the queue-limit.

Let’s consider a more comprehensive example of how to calculate default queue-limits. Consider the following hierarchical policy-map attached to a GigabitEthernet Interface:

```
policy-map child
  class voice
    priority
    police cir 10m
  class mission-critical
    bandwidth 50000
policy-map parent
  class class-default
    shape average 100m
    service-policy child
interface GigabitEthernet1/0/0
  service-policy out parent
```

For completeness, the scheduling hierarchy for this policy-map would look as follows:



The child policy-map has three queuing classes: voice, mission-critical, and class-default. Let's examine each in turn:

The voice queue is a priority queue so queue-limit will default to 512 packets.

The mission-critical queue has the **bandwidth** command configured with a rate of 50 Mbps so the visible bandwidth will be 50 Mbps (refer to the table above). As this is an Ethernet-type interface the MTU is 1500 bytes:

queue-limit = 50 Mbps / 8 bits * .05 sec / 1500 bytes = 208 packets

Although the implicit class-default has no queuing command configured, the implicit excess weight is equivalent to configuring **bandwidth remaining ratio 1**. This means that class-default will inherit its visible bandwidth from the parent (refer to the table above). At the parent, notice the shape configured with a value of 100 Mbps. The visible bandwidth for class-default in the child is therefore 100 Mbps and as before the MTU for the interface type is 1500 bytes:

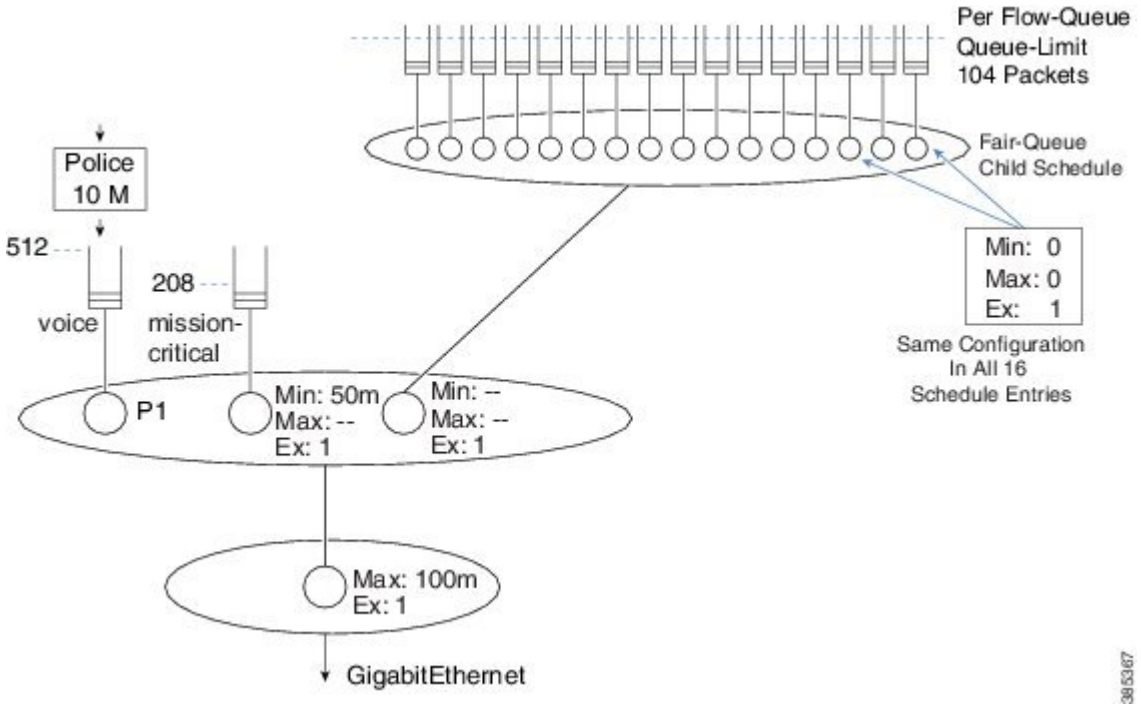
queue-limit = 100 Mbps / 8 bits * .05 sec / 1500 bytes = 416 packets

When Fair-Queue is Configured

In flow-based fair queuing we introduced flow-based fair queuing, where we configure 16 individual flow queues for a class and each flow-queue is configured with the same queue-limit. By default, this limit is ¼ of what is calculated based on the visible bandwidth of the class where the fair-queue feature is configured.

As an example, let's add fair-queue to class-default in the previous configuration example (see the asterisks):

```
policy-map child
  class voice
    priority
    police cir 10m
  class mission-critical
    bandwidth 50000
  class class-default                          ****
    fair-queue                                 ****
policy-map parent
  class class-default
    shape average 100m
    service-policy child
interface GigabitEthernet1/0/0
  service-policy out parent
```



Previously we had calculated queue-limit for class-default to be 416 packets based on the visible bandwidth inherited from the shaper in the parent.

Because flow-based fair-queuing is configured, we create 16 flow queues for that one class. The queue-limit for each individual flow queue is set as 104 packets – ¼ of the 416 packets we calculated.

Changing Queue-Limits

As stated previously, the default queue-limits set by the platform should apply to the majority of users but occasionally tuning them might be required.

Why and When to Change Queue-Limits

Three general situations necessitate tuning queue-limits: OOR drops, bursty traffic leading to tail drops, and latency issues.

When you observe OOR drops, you might need to reduce queue-limits to avoid the situation. As we anticipate that each *bandwidth remaining queue* will inherit its visible bandwidth from the parent, OOR drops may occur when many such queues are created. Additionally, changing queue-limits to byte mode might grant more granular control over how much packet memory a given queue may consume.

Occasionally, we observe that the rate of a stream over time is less than the minimum service rate of a queue. Yet, packets are still tail dropped. You can experiment by dramatically increasing the queue-limit. If systemic oversubscription is the cause, you will tail drops no matter how large you make the queue-limit. If burstiness is causing the drops you should no longer see packet loss. A good starting point is to double the queue-limit – if drops are gone then try reduce to 1.5 times the original queue-limit. You want to find a point where drops are no longer seen but not use unreasonably large queue-limits that may in turn lead to OOR issues. Note that

schedule burstiness caused by mixing very low rates with high rates in the same schedule could also be a cause.

Finally, you might need to adjust queue-limits to avoid unreasonable latency if a queue were to become congested. If you have queues with a visible bandwidth of less than roughly 15Mbps they will be assigned the default minimum queue-limit of 64 packets. If you add multiple queues to low speed interfaces, the minimum guaranteed service rates for those queues can become particularly low. Changing queue-limits to byte mode can be a good choice here.

For QoS Queue

You can use the **queue-limit** command to modify queue limits in any class containing a queuing action (bandwidth, bandwidth remaining, priority or shape). The queue limit may be specified in packets (default), bytes, or time. (We will review an example of each.) Here is an example of setting the limit in packet mode:

```
policy-map packet-mode-example
  class critical-data
    bandwidth percent 50
    queue-limit 2000
```

When you use the **queue-limit** command with the byte option, the second option, you are changing the queue's mode from packet to byte (as discussed previously). For the change to execute, you will need to remove and reattach the policy-map (or save configuration and reload the router). If you want to specify WRED thresholds in bytes you must first use the **queue-limit** command to change the mode of the queue to bytes:

```
policy-map byte-mode-example
  class critical-data
    bandwidth percent 50
    queue-limit 5000 bytes
```

Note If you attempt to change the queue-limit mode while a policy is attached to an interface, you will see an error message:

```
queue-limit 5000 bytes
Runtime changing queue-limit unit is not supported,please remove service-policy first
```

The third option is to specify the queue limit in time (milliseconds). Actually, the hardware only supports units in either packets or bytes. When you specify the unit in milliseconds the router will convert this to bytes; you are effectively changing the mode to byte. The router will use the visible bandwidth of the class (see When QoS is Configured, on page 297).

```
policy-map time-mode-example
  class critical-data
    shape average 20m
    queue-limit 50 ms
```

In this example the visible bandwidth of the queue is 20 Mbits/sec (2.5 Mbytes/sec). In 50mS at a rate of 2.5 Mbytes per/sec, you generate 125000 bytes of data (0.05s*2.5 Mbps). Therefore, in this example, we would set queue-limit at 125000 bytes. You can verify the value calculated in the output of the **show policy-map interface** command.

For Interface Default Queue

You cannot directly change the queue-limit for an interface that does not have an attached QoS policy. In IOS classic, the **hold-queue** command achieved this. In IOS XE, the hold queue exists within the IOSd daemon but is meaningless in the regular packet forwarding path. However, adjusting the hold-queue still has meaning for packets punted to IOSd, provided you have a topology with a very large number of routing peers and require more buffering within IOSd to handle simultaneous updates from all those peers.

To change the queue limit for the interface default queue, you can attach a simple policy-map with just class-default:

```
policy-map modify-interface-queue
  class class-default
    queue-limit 100 ms
!
interface gigabitethernet1/0/0
  service-policy out modify-interface-queue
```

WRED

WRED is a feature that monitors queue utilization. Under congestion and to alleviate further congestion, it randomly drops packets signaling endpoints to reduce their transmission rate.

Relience on Elasticity of IP Flows

WRED relies on the *elasticity of many IP flows*, where *elastic* describes flows that increase and reduce their send rate when the receiver detects missing packets. A very good example of elasticity is TCP. It starts slowly then increases the sender's *congestion window* (amount of outstanding unacknowledged traffic allowed) until either it reaches the maximum *receiver's receive window size* or it loses packets in the network. If the latter, it switches to a *congestion avoidance algorithm*, attempting to settle at the maximum congestion window size achievable without losing packets (see RFC 5681 for further details).

Another good example of elastic traffic is video (consider your favorite video streaming application). After starting the video, you typically observe that video quality improves as the rate increases. The rate continues to increase until the application discerns the capacity of the network. When it detects drops in the network it will recede, delivering the highest quality possible given the prevailing network conditions.

The How of WRED

With the diagram above, we visualize how WRED operates.

Senders behind our router send traffic to receivers elsewhere in the network. WRED is configured on the link (interface) connecting the router to the network. When the sum of the send rates of all the flows exceeds the link capacity we observe packets backing up in the queue configured for that interface.

In the section Tail Drop, on page 291, we described a tail-drops threshold for a queue. WRED uses a lower (minimum) threshold to determine when congestion is occurring. When the queue-depth reaches this threshold, we randomly drop packets rather than enqueue them, despite the queue spaces that are still available. The random nature of the drops ensures that we only drop packets from a small number of flows.

Let's say that you initially drop a single packet from Flow 1. TCP (or whatever elastic transport mechanism) will detect that drop and reduce the send rate of that flow. If by doing so, the link rate now exceeds the

aggregate send rate, then the queue depth will start to fall. If the queue depth falls below the WRED minimum threshold then WRED will cease dropping packets.

If the aggregate send rate still exceeds the link rate then the queue depth will continue to increase and WRED will continue to randomly drop packets. What if we now drop a packet from Flow 4, both Flows 1 and 4 are now backed off. This process continues until enough streams back off to alleviate the congestion.

The random element of WRED ensures that not all flows back off at the same time. If they did, they would likely try to increase their send rates again at the same time, resulting in a saw tooth effect where in synchrony, all senders reduce and increase their send rates. By randomly selecting packets to drop we randomly signal that different flows should back off at different times.

Average Queue Depth

In the previous discussion of WRED, we described random drops occurring when the queue depth crossed a predetermined threshold. Actually, we use a dampened average of the queue depth rather than the *instantaneous queue-depth*, which we use for the tail drop check and *average queue depth* for WRED.

Surely, this is no surprise: internet traffic is bursty. If we used instantaneous queue depth to monitor congestion we might drop packets in haste and so respond to normal bursts in traffic rather than to real congestion.

To determine how changes in average queue depth are dampened, we use the *WRED Exponential Weighting Constant*. The router will remember the current value of average queue depth. Whenever a packet reaches the enqueue stage, we examine the instantaneous queue depth and recalculate the average queue depth. The formula to calculate the new value of average queue depth is as follows:

Avg = OldAvg + (Instantaneous-OldAvg)/2^exp-weighting-constant

where Avg is the average queue depth calculated at the current enqueue time; Instantaneous, the current queue depth; and OldAvg, the previously calculated average that we remembered since the last enqueue.

For example, if the OldAvg is 12.0 packets, the Instantaneous is 14 packets (observed upon enqueuing a packet), and exp-weighting-constant is 6 (the default for packet mode WRED on the ASR 1000 Router), the Avg would be:

Avg = 12 + (14 – 12) / 2^6 = 12 + .03125 = **12.03125**

Note exp-weighting-constant = 9 if the queue is run in byte mode.

Later, we enqueue another packet. If one packet was transmitted from the head of the queue in the interim, the instantaneous queue depth would remain 14. Now, the calculation of AVG yields:

Avg = 12.03125 + (14 – 12.03125)/2^6 = 12.03125 + 0.0308 = **12.06201**

The example shows that the average queue depth is dampened. The instantaneous queue depth can grow considerably beyond the average. Consequently, the WRED max threshold is always considerably less than the queue limit. The example also illustrates the time necessary for the average to converge on the instantaneous, even if the queue depth stays consistent for some time. This dampening of average queue depth is how WRED avoids reacting to regular microbursts in traffic.

Only with a PHd in voodoo mathematics, should you consider changing the value of EWC. It is a "true geek knob" that should be avoided. For completeness only and not to encourage, here is the code change the EWC:

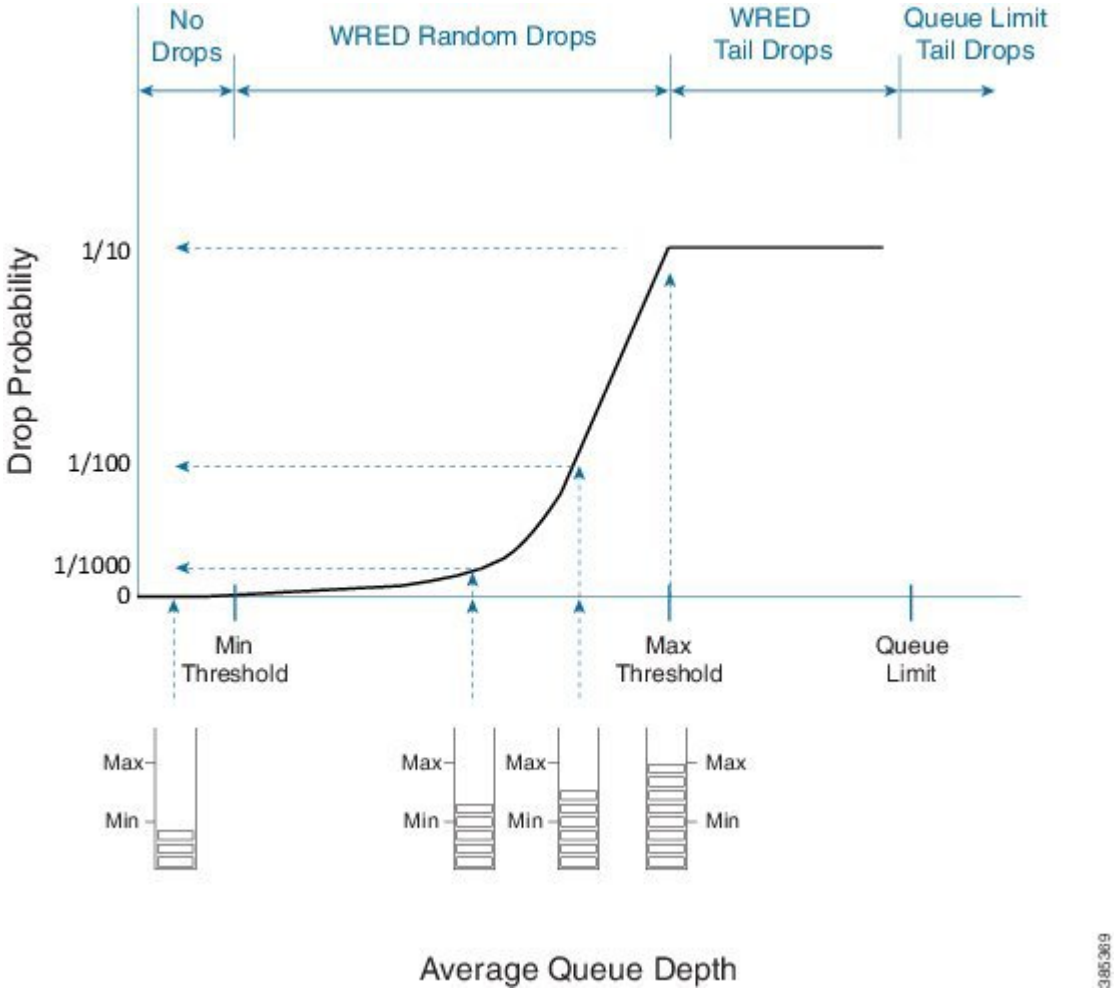
```
policy-map ewc-example
  class class-default
    random-detect
    random-detect exponential-weighting-constant 5
```

WRED Threshholds and Drop Curves

WRED drop decisions are driven by the average queue depth calculated upon enqueue.

When configuring WRED, we set a Minimum threshold, a Maximum threshold, and a Drop Probability for each precedence value (or DSCP, discard-class, etc.).

The following diagram shows the drop curve for a sample precedence value.



If the average queue depth is calculated to a value less than the WRED minimum threshold we are in a range where WRED will not drop any packets.

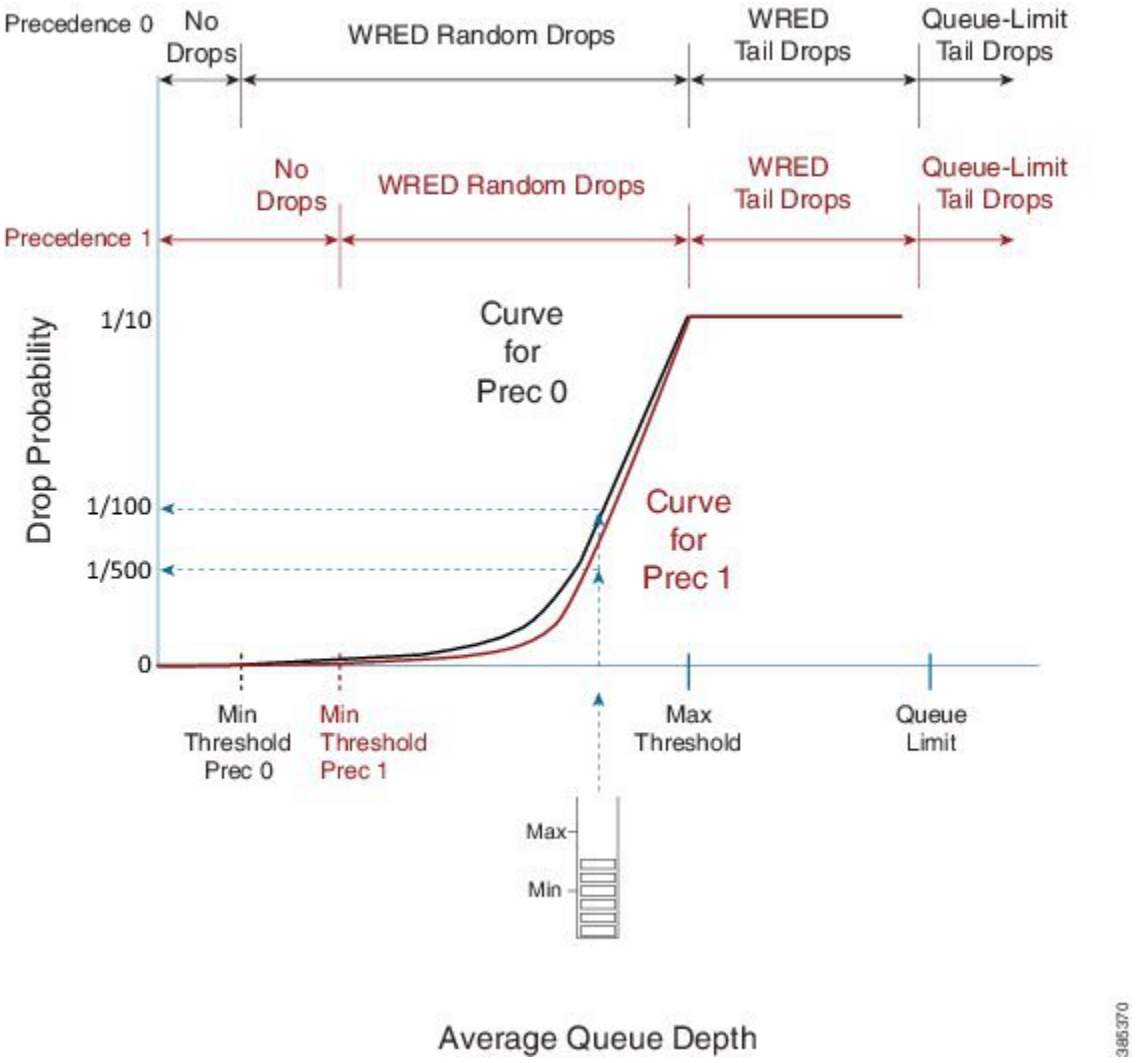
Between the Min and Max thresholds we are in the WRED random drop zone. Observe the exponential rise in drop probability from zero at the Min threshold to the configured WRED drop probability (defaults to 1 in 10 packets) at the Max threshold. The exponential nature of the curve means WRED will drop a very small number of packets when the average queue depth approximates the minimum threshold. As the illustration

as reflects, the average queue depth and drop probability increase in tandem. Knowing the average queue depth, we know the associated drop probability. Then, we decide whether to drop the packet or enqueue it.

If the WRED average queue depth is calculated to exceed the maximum threshold then the packet will experience a *WRED tail drop*. This differs slightly from the queue-limit tail drop - it is driven by average rather than instantaneous queue depth. If the instantaneous queue depth reaches the class's queue limit then the drop would be recorded as a queue-limit tail drop as opposed to a WRED tail drop.

Please note that the 'W' in WRED stands for *weighted* (some traffic may be dropped more aggressively than others).

Here, we show how to use multiple drop curves:



If a packet with IP precedence 0 arrives, the router will apply the black curve to calculate the drop probability. In the example the drop probability is calculated as 1 in 100 packets.

If a packet with IP precedence 1 arrives, we apply the mauve-colored curve. For the same average queue depth we would now see a drop probability of just 1 in 500.

Notice how the default maximum threshold is the same for each precedence value. The difference in WRED minimum threshold for each precedence value means that we will start dropping precedence 0 traffic before we drop any other traffic. Moreover, we will be more aggressive in dropping that traffic for a given queue depth in the random drop zone.

Note When you configure WRED, for each drop curve, the router will pick appropriate values for Min threshold, Max threshold and Drop Probability. Those values depend on the configured queue limit thereby accounting for the interface speed. We strongly recommend that you use the default values unless you fully understand the implications of any change.

WRED - Changing Drop Curves

Regardless of the WRED mode, you can tune any individual drop curve. Using the same command you may change the Minimum Threshold, the Maximum Threshold or the Drop Probability at Maximum Threshold for that drop curve. With the minimum and maximum thresholds and the drop probability, a router can construct the exponential curve it needs to determine drop probability for any average queue depth. Tuning WRED parameters is not typical; do not attempt unless you have a thorough understanding of how tuning will impact applications in that class. The default values should suffice for the vast majority of use cases.

If you decide to tune WRED drop curves, you have the option to specify thresholds in packets (default), bytes or time. The queue-limit must be configured in the chosen unit before you add WRED configuration to the class and only when the queue is already running in the desired mode can you change thresholds in that unit. Moreover, you can only change the curve for a particular DSCP, precedence or discard-class value provided WRED is operating in that mode.

Recall that the drop probability is an integer number. If the average queue limit is at the maximum threshold, a packet has a *1 in that integer value chance* of being dropped. For example, if the drop probability is 20, a 1 in 20 (5%) chance exists for a packet to dropped by WRED.

The command to change a drop curve is **random-detect [dscp|precedence|discard-class]** *value min-threshold max-threshold drop-probability*, as illustrated here:

```
policy-map tuneprecedence
  class bulk-data
    bandwidth remaining percent 30
    random-detect
    random-detect precedence 1 1301 2083 10
```

Running the queue in packet mode (the default) and WRED in precedence mode (also the default), I decide against differentiation in the minimum threshold for precedence 1 and 2. I change the curve for precedence 1, setting the minimum threshold to 1301, the maximum threshold to 2083 and the drop probability at max threshold to 1 in 10 packets:

random-detect precedence 1 1301 2083 10

As always, we can verify the configuration with the **show policy-map interface** command:

```
show policy-map interface g1/0/0
 GigabitEthernet1/0/0

  Service-policy output: tuneprecedence

    Class-map: bulk-data (match-all)
```

```
  0 packets, 0 bytes
  5 minute offered rate 0000 bps, drop rate 0000 bps
  Match: access-group name bulkdata
  Queueing
  queue limit 4166 packets
  (queue depth/total drops/no-buffer drops) 0/0/0
  (pkts output/bytes output) 0/0
  bandwidth remaining 30%
    Exp-weight-constant: 4 (1/16)
    Mean queue depth: 1086 packets
    class  Transmitted   Random drop   Tail drop   Minimum   Maximum   Mark
           pkts/bytes    pkts/bytes    pkts/bytes  thresh    thresh    prob

    0         0/0           0/0           0/0       1041      2083      1/10
    1         0/0           0/0           0/0       1301      2083      1/10
    2         0/0           0/0           0/0       1301      2083      1/10
    3         0/0           0/0           0/0       1431      2083      1/10
    4         0/0           0/0           0/0       1561      2083      1/10
    5         0/0           0/0           0/0       1691      2083      1/10
    6         0/0           0/0           0/0       1821      2083      1/10
    7         0/0           0/0           0/0       1951      2083      1/10
```

Notice the new values we set for precedence 1.

What if we change the thresholds for a queue that is running in time-based mode where WRED is running in DSCP mode? In particular, we want the minimum threshold of af21 to exceed that of af11. The configuration would appear as follows:

```
policy-map tunedscp
  class bulk-data
    bandwidth remaining percent 30
    queue-limit 50 ms
    random-detect dscp-based
    random-detect dscp af21 22 ms 25 ms 10
```

Looking at the output of **show policy-map interface** we verify the configuration:

```
show policy-map interface g1/0/0
 GigabitEthernet1/0/0

  Service-policy output: tunedscp

    Class-map: bulk-data (match-all)
      148826 packets, 223239000 bytes
      5 minute offered rate 2358000 bps, drop rate 0000 bps
      Match: access-group name bulkdata
      Queueing
      queue limit 50 ms/ 6250000 bytes
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 148826/223239000
      bandwidth remaining 30%

        Exp-weight-constant: 9 (1/512)
        Mean queue depth: 0 ms/ 992 bytes
        dscp    Transmitted     Random drop  Tail drop   Minimum    Maximum     Mark
                pkts/bytes      pkts/bytes   pkts/bytes  thresh     thresh      prob
                                                         ms/bytes   ms/bytes
        af11    96498/144747000   0/0          0/0       21/2734375 25/3125000  1/10
        af21    52328/78492000    0/0          0/0       22/2750000 25/3125000  1/10
```

With DSCP-based WRED we will only show curve statistics for DSCP values that have been observed within that class (refer to Mode: Precedence, DSCP, and Discard-Class, on page 310).

WRED Max Thresholds for Priority Enqueue

In the WRED - Changing Drop Curves, on page 306, we showed how to tune the minimum threshold of WRED curves. Another option is to modify the maximum threshold. When you do so with different thresholds for different DSCP values you can effectively claim that under congestion we always drop one type of traffic.

Let's use af11 to designate in-contract bulk data traffic and af12 to designate out-of-contract bulk data traffic? Under congestion, we want to always provide preferential treatment to af11 over af12. If we specify a lower WRED maximum threshold for af12 we could drop this traffic while still enqueuing af11.

In the following configuration, we change the maximum threshold for af12 from the default of 624 packets (for this bandwidth) to 580 packets:

```
policy-map maxthreshold
  class bulk-data
    bandwidth percent 30
    random-detect dscp-based
    random-detect dscp af12 468 580 10
```

Let's verify the configuration:

```
show policy-map interface g1/0/0
 GigabitEthernet1/0/0

  Service-policy output: maxthreshold

    Class-map: bulk-data (match-all)
      359826 packets, 539739000 bytes
      5 minute offered rate 7208000 bps, drop rate 0000 bps
      Match: access-group name bulkdata
      Queueing
      queue limit 1249 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 359826/539739000
      bandwidth 30% (300000 kbps)
        Exp-weight-constant: 4 (1/16)
        Mean queue depth: 0 packets
        dscp   Transmitted     Random drop   Tail drop    Minimum   Maximum   Mark
               pkts/bytes      pkts/bytes    pkts/bytes   thresh    thresh    prob

        af11  154689/232033500    0/0           0/0        546       624      1/10
        af12  205137/307705500    0/0           0/0        468       580      1/10
```

Looking at the configuration you can see that if the average queue depth exceeds 580 packets, all af12 packets would be *WRED tail dropped* but we would still enqueue af11 packets.

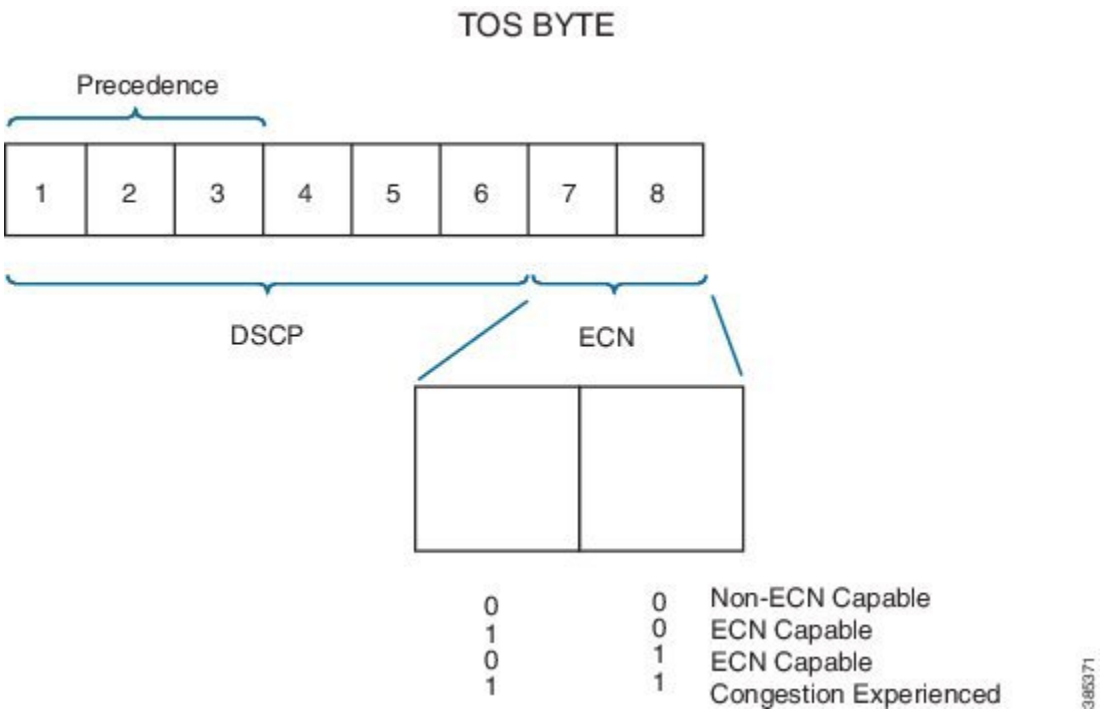
Be alert when modifying maximum thresholds to ensure that behavior is as expected. Here, if congestion persists and the average queue depth remains above 580 packets, then we would totally starve af12 traffic of any service during persistant congestion.

ECN - Explicit Congestion Notification

Explicit Congestion Notification (ECN) is an extension to the IP protocol where the network can mark a packet to signal congestion to the endpoints rather than drop packets early to signal congestion. Upon receiving such a packet, the endpoint echoes that congestion notification back to the sender.

Note ECN mode must be explicitly enabled in WRED.

To understand ECN you should first grasp the TOS byte in the IP header. Originally it was used to carry IP precedence bits in the 3 most significant bits, and more recently, to carry the DSCP codepoint in the 6 most significant bits of this byte. We define the remaining 2 bits in RFC3168 as ECN bits.



When WRED is configured in ECN mode it will look at the ECN bits before dropping a packet. If these bits are both set to zero, the router assumes the endpoints are ECN incapable and WRED will drop the packet to signal congestion is occurring.

If either of the ECN bits is set to 1, the router assumes that the endpoint is ECN capable and can mark congestion experienced rather than dropping the packet by setting both ECN bits to 1. The endpoints must signal a transport is ECN capable only if the upper layer protocol is elastic in nature.

Note The router will only look at the ECN bits when determining whether to mark or drop a packet.

The following is an example of configuring WRED in ECN mode:

```
policy-map ecn-example
  class bulk-data
    bandwidth remaining percent 30
```

```
random-detect dscp-based
random-detect ecn
```

Mode: Precedence, DSCP, and Discard-Class

WRED Precedence Mode

In WRED Threshholds and Drop Curves, on page 304 we describe drop curves.

When you enable WRED the default is to run in *precedence mode* and create 8 distinct drop curves, one for each valid precedence value. The default minimum threshold increases with the precedence value. The impact is that precedence 0 will start dropping earlier and more aggressively than precedence 1, precedence 1 earlier and more aggressively then precedence 2, etc. The same default maximum threshold and drop probability is configured for each curve.

When a packet arrives the IP precedence bits determine which curve we use to find the appropriate drop probability. If the packet is not "IP," we use the precedence 0 drop curve. If the packet is MPLS encapsulated then the EXP bits are treated as precedence bits and determine the appropriate drop curve.

In the following example we enable WRED in precedence mode. Observe that WRED must reside in a class that has a queuing action (including class-default):

```
policy-map wred-precedence-example
  class bulk-data
     bandwidth remaining percent 30
     random-detect
     random-detect precedence-based
```

In this example, we use **bandwidth remaining** command as the queuing action. The **random-detect** command enables WRED in the class bulk-data and the **random-detect precedence-mode** command tells WRED to operate in precedence mode.

Note The **random-detect precedence-mode** command is optional as the default mode for WRED is precedence-based.

As with all QoS features the **show policy-map interface** command is the primary means to verify your configuration:

```
show policy-map int g1/0/0
 GigabitEthernet1/0/0

  Service-policy output: wred-precedence-example

   Class-map: bulk-data (match-all)
    6468334 packets, 9702501000 bytes
    5 minute offered rate 204108000 bps, drop rate 0000 bps
    Match: access-group name bulkdata
    Queueing
    queue limit 4166 packets
    (queue depth/total drops/no-buffer drops) 1308/0/0
    (pkts output/bytes output) 6468335/9702502500
    bandwidth remaining 30%
     Exp-weight-constant: 4 (1/16)
     Mean queue depth: 1308 packets
     class     Transmitted       Random drop    Tail drop      Minimum    Maximum     Mark
```

```
            pkts/bytes          pkts/bytes      pkts/bytes      thresh      thresh      prob

  0             0/0               0/0             0/0           1041        2083        1/10

  1             0/0               0/0             0/0           1171        2083        1/10

  2             0/0               0/0             0/0           1301        2083        1/10

  3             0/0               0/0             0/0           1431        2083        1/10

  4       6468335/9702502500      0/0             0/0           1561        2083        1/10

  5             0/0               0/0             0/0           1691        2083        1/10

  6             0/0               0/0             0/0           1821        2083        1/10

  7             0/0               0/0             0/0           1951        2083        1/10
```

Notice how statistics and curve configuration values are displayed for each of the 8 drop curves that are created in precedence mode. The average queue-depth is less than the minimum threshold so no random drops are reported.

WRED DSCP Mode

The second option for configuring WRED is DSCP mode, where we create 64 unique curves.

Similar to precedence mode, any non-IP traffic will use the default (DSCP 0) curve. If MPLS traffic is seen, we treat the MPLS EXP bits as precedence values and select the curve accordingly (EXP 1 treated as DSCP CS1, EXP 2 as CS2, etc.).

Here is an example of configuring WRED in DSCP mode:

```
policy-map wred-dscp-example
  class bulk-data
    bandwidth remaining percent 30
    random-detect dscp-based
```

Here, we verify the configuration with the **show policy-map interface** command:

```
show policy-map int
 GigabitEthernet1/0/0

  Service-policy output: wred-dscp-example

    Class-map: bulk-data (match-all)
      5655668 packets, 8483502000 bytes
      5 minute offered rate 204245000 bps, drop rate 0000 bps
      Match: access-group name bulkdata
      Queueing
      queue limit 4166 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 5655669/8483503500
      bandwidth remaining 30%
       Exp-weight-constant: 4 (1/16)
       Mean queue depth: 1 packets
       dscp   Transmitted      Random drop   Tail drop    Minimum   Maximum   Mark
              pkts/bytes       pkts/bytes  pkts/bytes    thresh     thresh    prob
       af11 1205734/1808601000     0/0         0/0         1821      2083      1/10
```

```
  cs4  5270109/7905163500     0/0         0/0         1561      2083      1/10
```

Notice that we only display statistics and drop curve information for 2 DSCP values (af11 and cs4). In DSCP mode, 64 unique drop curves are configured and IOS will maintain statistics for all. However, it will only display information for drop curves that have actually observed traffic. In this example, we have observed only display traffic with DSCP af11 and cs4, hence the display.

WRED Discard-Class

Discard-class is an internal marking very similar in concept to qos-group. We can mark discard-class on ingress (and not on egress) as well as employ to select a WRED drop curve on egress.

Occasionally, the precedence or DSCP marking in a packet is unavailable for classification on an egress interface. A use-case is an MPLS-encapsulating router where we receive IP packets on the ingress interface and forward MPLS-encapsulated packets on the egress interface.

DSCP must be mapped into a smaller number of EXP values (6 bits in the DiffServ field vs 3-bit field in MPLS header) so some granularity is lost. Let's say af11 is used for in-contract and af12 for out-of-contract bulk data. On the egress interface the DSCP visibility is lost; af11 and af12 would probably be mapped into the same EXP. Now, what if we want to provide preferential treatment to af11 over af12 on the egress interface?

We could use WRED discard-class mode to achieve this. To do so, you will need to mark discard-class on ingress interfaces, as in the following sample policy:

```
policy-map mark-in-contract
  class bulk-data
    police cir 50000000 pir 100000000
    conform-action set-dscp-transmit af11
    conform-action set-mpls-exp-imposition-transmit 1
    conform-action set-discard-class-transmit 2
    exceed-action set-dscp-transmit af12
    exceed-action set-mpls-exp-imposition-transmit 1
    exceed-action set-discard-class-transmit 1
    violate-action drop
```

In this policy traffic adhering to the CIR is marked as in-contract:

```
    conform-action set-dscp-transmit af11
    conform-action set-mpls-exp-imposition-transmit 1
    conform-action set-discard-class-transmit 2           ****
```

Traffic between the CIR and PIR is marked as out-of-contract:

```
    exceed-action set-dscp-transmit af12
    exceed-action set-mpls-exp-imposition-transmit 1
    exceed-action set-discard-class-transmit 1            ****
```

Violating traffic is dropped.

Notice how the same EXP value will be set for conforming and exceeding traffic – it is all bulk data traffic and will use the same per-hop-behavior in the MPLS network. However, for in-contract and out-of-contract traffic we also mark distinct discard-classes (see the asterisks), which we use use on the egress interface to provide preferential treatment.

On the egress interface you would configure WRED in discard-class-based mode, as follows:

```
policy-map wred-discard-class-example
  class bulk-data
    bandwidth remaining percent 30
    random-detect discard-class-based
```

Looking at the output of **show policy-map interface** command you will see something like:

```
show policy-map int g1/0/0
 GigabitEthernet1/0/0

  Service-policy output: wred-discard-class-example

    Class-map: bulk-data (match-all)
      1500 packets, 1040000 bytes
      5 minute offered rate 51955000 bps, drop rate 0000 bps
      Match: access-group name bulkdata
      Queueing
      queue limit 4166 packets
      (queue depth/total drops/no-buffer drops) 943/0/0
      (pkts output/bytes output) 1500/1040000
      bandwidth remaining 30%
        Exp-weight-constant: 4 (1/16)
        Mean queue depth: 943 packets
        discard-class  Transmitted  Random drop   Tail drop   Minimum   Maximum  Mark
                       pkts/bytes   pkts/bytes    pkts/bytes  thresh    thresh   prob

        0                 0/0          0/0          0/0        1041      2083    1/10
        1               500/4000       0/0          0/0        1171      2083    1/10
        2              1000/1000000    0/0          0/0        1301      2083    1/10
        3                 0/0          0/0          0/0        1431      2083    1/10
        4                 0/0          0/0          0/0        1561      2083    1/10
        5                 0/0          0/0          0/0        1691      2083    1/10
        6                 0/0          0/0          0/0        1821      2083    1/10
        7                 0/0          0/0          0/0        1951      2083    1/10
```

Looking at the output you can see that 8 drop curves are created when you run WRED in discard-class mode. Referring to the configuration above, in-contract traffic is marked with discard-class 2 and out-of-contract traffic is marked with discard-class 1.

You can also see that the WRED curve for discard-class 1 has a lower minimum threshold. This means that under congestion, out-of-contract traffic will start dropping earlier and more aggressively than in-contract traffic.

Any traffic devoid of an explicitly-set discard-class is assumed to that does not have a discard-class explicitly set will be assumed to be discard-class 0.

Command Reference - random detect

Use the **random-detect** *options* command to enable and control operation of WRED, applying different options as below.

To enable WRED – use one of the following:
random-detect

Enable WRED in precedence mode.

random-detect precedence-based

Enable WRED in precedence mode.

random-detect dscp-based

Enable WRED in DSCP mode.

random-detect discard-class-based

Enable WRED in discard-class mode.

To tune WRED Drop Curve – use one of the following

random-detect precedence *value min-threshold max-threshold drop-probability*

Modify the drop curve for a particular precedence value

random-detect dscp *value min-threshold max-threshold drop-probability*

Modify the drop curve for a particular DSCP value

random-detect precedence *value min-threshold max-threshold drop-probability*

Modify the drop curve for a particular discard-class value. Note the min-threshold and max-threshold may be configured in packets (default), bytes or time. To use the units of bytes or time the queue must first be configured for that mode using the **queue-limit** command.

To change the WRED Exponential Weighting Constant

random-detect exponential-weighting-constant *value*

To enable Explicit Congestion Notification Support

random-detect ecn

Usage:

The **random-detect** command may be used in any queuing class configured with the **bandwidth**, **bandwidth remaining** or **shape**commands. This includes class-default which has an implicit bandwidth remaining value.

The ASR 1000 Series Aggregation Services Router has no queues in parent or grandparent levels of a scheduling hierarchy. So, the **random-detect** command is not supported in any class that contains a child queuing policy.

The default values for WRED minimum and maximum thresholds are proportional to the queue-limit for a class and therefore proportional to the expected service rate of the queue. Modifying WRED drop curves should not be undertaken unless you have a deep understanding on how changes will affect applications in that class.