

Applying QoS Features Using the MQC

- About, on page 1
- Cisco Modular QoS CLI, on page 1
- Create Class Maps, on page 2
- Create Policy-Maps, on page 3
- Attach the Policy-Map, on page 7
- Verify Operation of the QoS Policy, on page 7

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.