

10.33 MQC Classification and Marking

- Configure an outbound MQC policy on R4's Serial link to R5 per the following requirements:
 - WWW traffic from servers on VLAN 146 should be marked with an IP Precedence of 2
 - VoIP packets with UDP ports in the destination range of 16384 - 32767 and a Layer 3 packet size of 60 bytes should be marked with DSCP EF
 - ICMP packets larger than 1000 bytes should be dropped
 - All other packets that came from R4's connection to VLAN 146 with an IP precedence of 0 should be remarked with an IP precedence of 1
- Do not use an access-list to classify ICMP packets.

Configuration

```
R4:
ip access-list extended HTTP
 permit tcp 155.1.146.0 0.0.0.255 eq www any
!
ip access-list extended VOICE
 permit udp any any range 16384 32767
!
class-map HTTP
 match access-group name HTTP
!
class-map match-all LARGE_ICMP
 match protocol icmp
 match packet length min 1001
!
class-map match-all VOICE
 match access-group name VOICE
 match packet length min 60 max 60
!
class-map match-all SCAVENGER
 match input-interface FastEthernet0/1
 match ip precedence 0
!
policy-map SERIAL_LINK
 class VOICE
  set ip dscp ef
 class HTTP
  set ip precedence 2
 class LARGE_ICMP
  drop
 class SCAVENGER
  set ip precedence 1
!
interface Serial 0/1
 service-policy output SERIAL_LINK
```

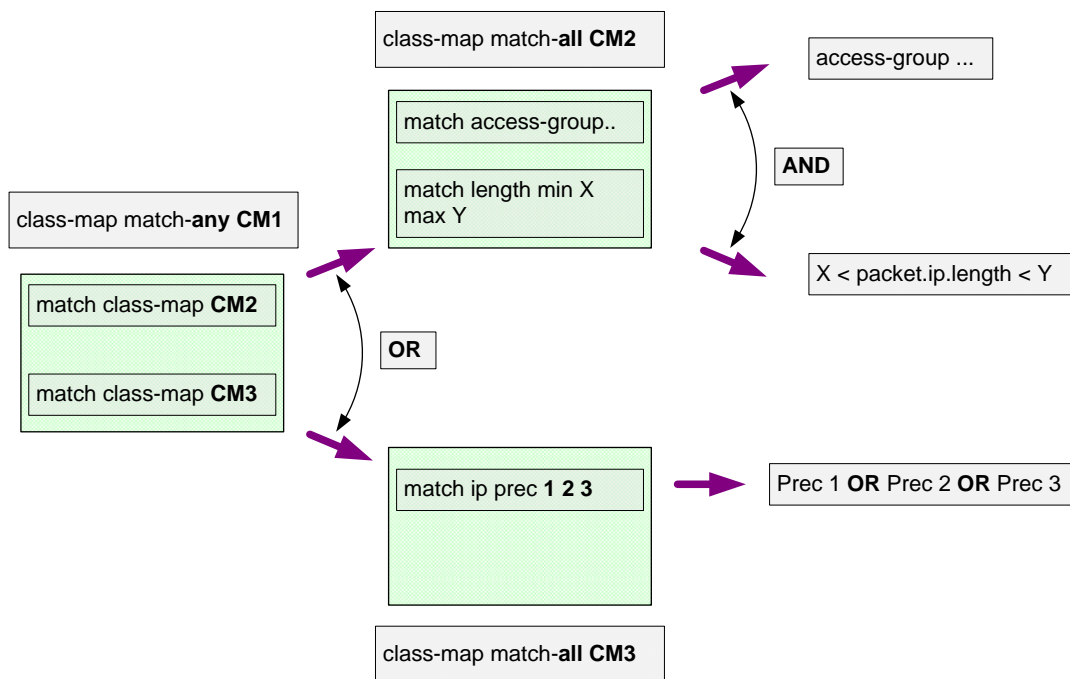
Verification

Note

The Modular Quality of Service Command Line Interface (MQC), also known as Class-Based Weighted Fair Queueing (CBWFQ), unifies all IOS QoS features under a single interface. MQC allows the implementation of a full suite of QoS tools, including classification, congestion management, traffic metering, marking, traffic shaping, and link efficiency. The main advantage of using the MQC over the legacy methods is that multiple QoS features can be applied to the same interface in the same direction. For example, with legacy QoS, you can't apply custom queueing and priority queueing at the same time, but with MQC you can.

Classification in MQC uses case sensitive *class-maps* (not to be confused with a frame-relay map-class) to group criteria. Each class-map performs a logical AND (match-all) or a logical OR (match-any) on its criteria. In other words in a match-all class-map, all matches must be TRUE in order for the class to be TRUE. Class-maps can be nested inside other maps to build complicated classification "AND-OR" logic gates. If multiple match criteria appear on the same line (e.g. match ip dscp, or match ip precedence), they are treated as a logical OR match.

MQC Classification Process



Different IOS versions and platforms support different matches in the class-map, but as a general rule the following classification criteria are supported:

- Named and numbered access-lists - allows matching of IP addresses, TCP/UDP ports, IP protocol numbers, etc.
- Layer 3 packet length
- Layer 2 addresses - source/destination MAC address, Frame-Relay DLCI, etc.
- Packet marking - Layer 2 CoS, Layer 3 DSCP/IP precedence, Frame Relay DE, ATM CLP, MPLS EXP, etc.
- Network-Based Application Recognition (NBAR)
- Inverse logical matching (logical NOT)

When you apply a logical NOT to a nested class-map or multiple criteria in a single line, *De Morgan's law* applies, where $\text{NOT}(X \text{ AND } Y) = (\text{NOT } X) \text{ OR } (\text{NOT } Y)$, while $\text{NOT}(X \text{ OR } Y) = (\text{NOT } X) \text{ AND } (\text{NOT } Y)$.

Once classification is configured in a class-map, actions are defined for the different classes in a case sensitive *policy-map*. A policy map is an *ordered* list of class-maps with their corresponding actions, similar to how a route-map works. The router matches packets entering/leaving the interface against all class-map entries in the respective input/output policy-map on the interface in a top-down fashion. This means that the first match in a class-map is used for classification, which implies that the order of the classes called in the policy-map is significant. The policy-map actions include marking, shaping, policing, assigning queue weight, compressing, etc. Any unclassified traffic in a policy-map falls into the *class-default* category, is covered in depth, along with the policy-map actions, in following sections.

Pitfall

Correct traffic flow classification within the class-map, and the correct order of operations in the policy-map, is key in the implementation of an MQC policy. In this task the question asks to classify traffic flows from web servers in VLAN 146, which means that they will be using *source port 80* in their responses to clients. Additionally the SCAVENGER class-map, which matches IP Precedence 0 traffic from VLAN 146, may overlap other traffic classes, such as the HTTP class, which makes it important that SCAVENGER is called last in the policy-map to match any un-classified traffic up to that point.

 **Note**

To verify this configuration first shutdown R5's Frame Relay link. Next, enable the HTTP server service on R1 and start transferring an IOS image from R1 to SW2, start an IP SLA jitter operation on R6 to source "voice-like" packets with the G.729 codec (60 bytes each), and finally send a large number of ICMP packets from R6 to R5, each larger than 1000 bytes.

```
R1:
ip http server
ip http path flash:
```

```
R5:
rtr responder
```

```
R6:
ip sla monitor 1
  type jitter dest-ipaddr 155.1.45.5 dest-port 16384 codec g729a
  timeout 1000
  frequency 1
!
ip sla monitor schedule 1 life forever start-time now
```

```
Rack1SW2#copy http://admin:cisco@155.1.146.1/c2600-adventerprisek9-
mz.124-10.bin null:
```

```
Rack1R6#ping 155.1.45.5 repeat 100 size 1004 timeout 0
```

```
Type escape sequence to abort.
Sending 100, 1004-byte ICMP Echos to 155.1.45.5, timeout is 0 seconds:
.....
Success rate is 0 percent (0/100)
```

Check the statistics to see the policy-map matches.

```
Rack1R4#show policy-map interface serial 0/1
Serial0/1
```

```
Service-policy output: SERIAL_LINK
```

```
Class-map: VOICE (match-all)
  2006 packets, 128384 bytes
  30 second offered rate 20000 bps, drop rate 0 bps
  Match: access-group name VOICE
  Match: packet length min 60 max 60
  QoS Set
    dscp ef
    Packets marked 2006

Class-map: HTTP (match-all)
  899 packets, 521420 bytes
  30 second offered rate 75000 bps, drop rate 0 bps
  Match: access-group name HTTP
  QoS Set
    precedence 2
    Packets marked 899

Class-map: LARGE_ICMP (match-all)
  100 packets, 100800 bytes
  30 second offered rate 24000 bps, drop rate 24000 bps
  Match: protocol icmp
  Match: packet length min 1001
  drop

Class-map: SCAVENGER (match-all)
  2 packets, 168 bytes
  30 second offered rate 0 bps, drop rate 0 bps
  Match: input-interface FastEthernet0/1
  Match: ip precedence 0
  QoS Set
    precedence 1
    Packets marked 2

Class-map: class-default (match-any)
  8 packets, 1568 bytes
  30 second offered rate 0 bps, drop rate 0 bps
  Match: any
```

Note that all MQC configurations use the same unified syntax for configuration and verification.

10.34 MQC Bandwidth Reservations and CBWFQ

- Modify R4's MQC policy to meet the following requirements:
 - Set the total size of the MQC queue to 512 buffers
 - The traffic flows for the web servers in VLAN 146 and the IP Precedence 0 packets from VLAN 146 should be guaranteed 32Kbps each
 - Limit the sizes of the FIFO queues for the HTTP and IP Precedence 0 traffic classes to 16 and 24 packets respectively
 - All other unmatched traffic in the policy should run WFQ
 - Dynamic flows in this WFQ should start dropping when they reach 32 packets in length

Configuration

```
R4:
interface Serial 0/1
  bandwidth 128
  max-reserved-bandwidth 100
  no fair-queue
  hold-queue 512 out
!
policy-map SERIAL_LINK
  class VOICE
  class HTTP
    bandwidth 32
    queue-limit 16
  class SCAVENGER
    bandwidth 32
    queue-limit 24
  class class-default
    fair-queue
    queue-limit 32
```

Verification

Note

Class-Based Weighted Fair Queueing (CBWFQ) is the MQC equivalent of a combination of the legacy interface level `fair-queue` command and the custom-queue. CBWFQ is used to reserve a *minimum* amount of bandwidth in the output queue for a particular traffic flow in the case of congestion. The `bandwidth` statement used in the policy-map for the class is used to compute the scheduling weight of the traffic flow, relative to the configured interface bandwidth.

The logic of CBWFQ is that during congestion, a class with a bandwidth reservation of *ClassBandwidth* will have at least a $\frac{\text{ClassBandwidth}}{\text{InterfaceBandwidth}}$ share of the total interface bandwidth. For this reason it is key that the interface level `bandwidth` statement is configured correctly, in respect to the access rate or guaranteed rate of the link in question whenever an MQC policy is applied. The `bandwidth` statement under the policy-map does not have a built-in policer, which means that if congestion is not occurring, the traffic flow matched by the class can use as much bandwidth as it wants. Based on this fact the CBWFQ reservation only becomes active once congestion occurs. Configuration of this feature is much more straightforward than the legacy custom-queue, as the bandwidth reserved is not based on a relative ratio like it is in the legacy version.

The feature is deemed Class-Based Weighted Fair Queueing, because the WFQ flow is manually defined through the MQC class-map. The weighting value for the flow is then defined through the `bandwidth` value under the respective policy-map. Once the policy-map is applied to the interface the entire software queue changes to CBWFQ. The CLI will not allow you to assign a service-policy with CBWFQ weights unless the interface is using FIFO queuing, which implies that CBWFQ is not compatible with any of the legacy queuing methods, such as custom queueing or priority queueing. These must be disabled explicitly before applying the service-policy.

By default, the sum of all configured bandwidth reservations cannot exceed 75% of the interface bandwidth value, and can be modified like the legacy RTP reservations with the `max-reserved-bandwidth` interface command. The system subtracts the bandwidth reserved for a class from the available interface bandwidth for the purpose of admission control. The default limits are designed to leave some best-effort bandwidth to the unclassified traffic, such as routing updates and layer 2 keepalives, which fall into the class-default of the policy-map.

Each class configured with a `bandwidth` statement under the policy-map has its own dedicated FIFO queue in the interface's CBWFQ conversation pool. The depth of each FIFO queue can be changed on a per-class basis with the `queue-limit` command. The overall WFQ settings, such as the CDT and the total queue size, can be set using the `queue-limit` under class-default, and the `hold-queue out` command at the interface level.

Recall that with legacy WFQ, flows are classified automatically. The scheduler shares the bandwidth in proportion to the flow's IP precedence value, normalized against the sum of other flows' precedences. Specifically if there are N active flows, and flow i has IP Precedence $IPP(i)$, then any flow k is guaranteed a share as:

$$\text{Share}(k) = (IPP(k)+1)/(IPP(1)+IPP(2)...+IPP(N)+N)*100\%$$

Note that each precedence value is shifted by 1 to ensure non-zero values. Since WFQ implements max-min sharing, any flow may claim unused interface bandwidth. Finally, the implementation uses a computation "weight" value based on the formula:

$$\text{Weight}(i) = 32384/(IPP(i)+1) \quad \text{[Formula 1]}$$

This allows for the flow with higher IP precedence to have lower computation weight, and larger share of bandwidth. Remember that computation weights are inversely proportional to the actual weights you use to compute the share of the bandwidth, as the bandwidth is shared in the proportion $[1/\text{Weight}(1):1/(\text{Weight}(2)... :1/\text{Weight}(N)]$. For example, WFQ assigns a weight of 4048 to a flow with an IP precedence of 7, which is the maximum IPP value. This is an important fact that will be referenced again later.

With the new MQC format, CBWFQ retains the above computations for *unclassified* traffic only. Traffic flows that are explicitly matched in a class-map and have a bandwidth reservation configured are treated differently. To start, this class uses a separate flow queue with its own limits. Secondly, this flow is assigned a computation weight based on the following formula:

$$\text{Weight}(i) = \text{Const} * \text{InterfaceBandwidth} / \text{Bandwidth}(i) \quad \text{[Formula 2]}$$

Here $\text{Bandwidth}(i)$ is the value configured under the respective class using the `bandwidth` keyword, and $\text{InterfaceBandwidth}$ is the value configured under the main interface using the `bandwidth` keyword. The value of the constant Const depends on the number of flow queues active in CBWFQ, and varies from 1 to 64. The implications of the above formula's calculations are as follows.

Almost any manually configured class has a weight value significantly lower than any automatically classified flow. Recall that IP Precedence 7, the best value, has a best possible weight of 4048 for a dynamic conversation. However for a manually defined class, even in the worst case, *Const* equals 64 (e.g. with 32 flow-queues), the ratio of *interfaceBandwidth/Bandwidth* should be less than 0.02 to be comparable to 4048. This means that a class should have less than a 2% reservation of the interface's bandwidth, which is rare. Furthermore since any user-defined class has its own separate FIFO queue, we can conclude that CBWFQ isolates it from congestive discard drops performed across dynamic WFQ queues. Instead, each class queue is FIFO with a tail-drop discipline by default.

The next question is how CBWFQ shares bandwidth between the user-defined classes. Looking at the weight calculation we can assume that each class has a relative bandwidth share of

Share(i) = Bandwidth(i)/InterfaceBandwidth.

Recall that by default, the sum of all *Bandwidth(i)* reservation is no more than 75% of *InterfaceBandwidth*, due to the **max-reserved-bandwidth** defaults. So what happens with the remaining 25%? This is where class-default comes in.

As soon as the **bandwidth** keyword is specified under any user-defined class, the interface queue turns into CBWFQ. This means any unmatched flows that fall back into class-default are scheduled using dynamic WFQ weights. This means that automatic classification occurs, along with precedence-based weight assignment and sharing of the single buffer space of WFQ. This behavior is default, even if you did not configure **fair-queue** under the class-default.

In addition to dynamic flow queues, WFQ and CBWFQ both have the concept of Link Queues. The system uses these queues for critical control plane traffic, such as routing updates and layer 2 keepalives. Each Link Queue has computation weight of 1024, which is better than any dynamic queue's weight. By default, CBWFQ matches all control plane traffic to class-default, so it is reasonable to assume that some bandwidth (25% by default) is saved for those flows. This safeguard measure prevents user-defined classes from oversubscribing the interface bandwidth, and does not let the other important queues starve.

If you want to disable fair-queue for unclassified packets, an explicit **bandwidth** value for the class-default can be configured, which turns it into a single FIFO queue. For example, the following code snippet disables fair-queue for unclassified traffic, and gives it a static weight as relative to 96Kbps:

```
policy-map TEST
  class class-default
    bandwidth 96
```

Moreover if you do not define any classes other than class-default, and class-default has a bandwidth value defined, the entire interface queue essentially becomes a FIFO queue.

The following tables illustrate the weight values that are used for CBWFQ. Here N is the constant that defines the number of dynamic WFQ queues. Their number is always a power of 2, and equals 2^N . Normally the IOS automatically calculates this based on the interface's bandwidth value, but it can be manually specified with the **fair-queue** command in class-default.

Conversation Numbers	CBWFQ Weight	Description
Below 2^N	$32384/(IPP+1)$	Used for automatic classification of dynamic WFQ Queues. Configurable via fair-queue command under "class-default".
$2^N \dots 2^{N+7}$	1024	Link Queues. There are 8 conversations used to queue routing updates (marked as <i>PAK_PRIORITY</i> internally) and Layer 2 keepalives.
2^{N+8}	0	Priority queue which maps directly to legacy WFQ IP RTP Priority. Typically used for VoIP bearer traffic. CBWFQ polices this flow using a configurable token bucket procedure to ensure it does not starve other queues.
Above 2^{N+8}	Const*IntfBW/ClassBW OR RSVP flow weight	These conversations are used for user-defined traffic classes. Each class has its own FIFO queue and configurable queue depth. In addition, RSVP flows use the same range of conversation numbers.

The "N" constant	Number of dynamic Flows	CBWFQ constant "Const"
4	16	64
5	32	64
6	64	57
7	128	30
8	256	16
9	512	8
10	1024	4
11	2048	2
12	4096	1

In summary, the key point about CBWFQ is that it uses the same scheduling logic as the legacy WFQ, but user-configurable classes have a special low-weight, making them more important than any dynamic conversation. Thus, user-defined classes always get the guaranteed proportion of bandwidth, while flows using class-default may starve in the case of congestion.

For verification of this task shut down R5's Frame Relay link to force the traffic from VLAN 146 to the hosts behind R4 to take the path across the serial link. R6 should send IP SLA jitter probes to R5 across the serial link; this flow simulates voice traffic and has no explicit bandwidth set under the class. Next, configure R1 as an HTTP server, and set SW2 to transfer the IOS image from R1, oversubscribing the serial link. Lastly, generate a flow of ICMP packets from R6 to R5, simulating the SCAVENGER class traffic.

```
R1:
ip http server
ip http path flash:
```

```
R4:
interface Serial 0/1
  load-interval 30
```

```
R5:
rtr responder
!
interface Serial 0/0
  shutdown
```

```
R6:
ip sla monitor 1
  type jitter dest-ipaddr 155.1.45.5 dest-port 16384 codec g729a
  timeout 1000
  frequency 1
!
ip sla monitor schedule 1 life forever start-time now
```

```
Rack1SW2#copy http://admin:cisco@155.1.146.1/c2600-adventerprisek9-
mz.124-10.bin null:
```

```
Rack1R6#ping 155.1.45.5 repeat 100000 size 150
```

Type escape sequence to abort.

Sending 100000, 150-byte ICMP Echos to 155.1.45.5, timeout is 2 seconds:

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

Check the policy-map matches and observe the measured traffic rates. Note that the load-interval is set to 30 seconds to ensure more adaptive response to changes in traffic patterns.

```
Rack1R4#show policy-map interface serial 0/1
Serial0/1
Service-policy output: SERIAL_LINK

Class-map: VOICE (match-all)
 173240 packets, 11087360 bytes
 30 second offered rate 23000 bps, drop rate 0 bps
Match: access-group name VOICE
Match: packet length min 60 max 60
QoS Set
  dscp ef
  Packets marked 173241

Class-map: HTTP (match-all)
 45477 packets, 26371423 bytes
 30 second offered rate 104000 bps, drop rate 0 bps
Match: access-group name HTTP
QoS Set
  precedence 2
  Packets marked 45477
Queueing
  Output Queue: Conversation 41
  Bandwidth 32 (kbps)Max Threshold 16 (packets)
  (pkts matched/bytes matched) 6020/3491600
  (depth/total drops/no-buffer drops) 6/0/0

Class-map: LARGE_ICMP (match-all)
 0 packets, 0 bytes
 30 second offered rate 0 bps, drop rate 0 bps
Match: protocol icmp
Match: packet length min 1001
drop

Class-map: SCAVENGER (match-all)
 38429 packets, 5905886 bytes
 30 second offered rate 15000 bps, drop rate 0 bps
Match: input-interface FastEthernet0/1
Match: ip precedence 0
QoS Set
  precedence 1
  Packets marked 104852
Queueing
  Output Queue: Conversation 42
  Bandwidth 32 (kbps)Max Threshold 24 (packets)
  (pkts matched/bytes matched) 2286/351274
  (depth/total drops/no-buffer drops) 0/0/0

Class-map: class-default (match-any)
 908 packets, 205544 bytes
 30 second offered rate 0 bps, drop rate 18000 bps
Match: any
Queueing
  Flow Based Fair Queueing
  Maximum Number of Hashed Queues 32
  (total queued/total drops/no-buffer drops) 64/9916/0
```

Note that the conversation numbers for the user-defined classes are 41 and 42, which means that there are 32 dynamic queues plus 8 link queues. The IOS picked the number of dynamic queues automatically based on interface bandwidth of 128Kbps.

Next, the class for VOICE traffic demonstrates a measured bitrate close to 24Kbps. However, we also see the high drop rate under class-default, along with its queue stats. Since the VOICE class has no weight of its own, CBWFQ assigns it to a dynamic queue.

Two other user-configured classes (HTTP and SCAVENGER) have equal bandwidth weights configured, but the SCAVENGER class is not consuming all of its allocated bandwidth. This is why the HTTP class uses the unclaimed resources. Finally, since both user configurable classes have better weights than the WFQ dynamic queues, the VOICE traffic cannot get enough bandwidth. Thus, CBWFQ drops most of the VOICE class traffic.

Now let's see the CBWFQ queue contents:

Rack1R4#show queueing interface serial 0/1

```
Interface Serial0/1 queueing strategy: fair
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 11322
  Queueing strategy: Class-based queueing
  Output queue: 12/512/32/0 (size/max total/threshold/drops)
    Conversations 3/4/32 (active/max active/max total)
    Reserved Conversations 3/3 (allocated/max allocated)
    Available Bandwidth 40 kilobits/sec

(depth/weight/total drops/no-buffer drops/interleaves) 6/256/0/0/0
Conversation 41, linktype: ip, length: 580
source: 155.1.146.1, destination: 155.1.58.8, id: 0x74CD, ttl: 254,
TOS: 64 prot: 6, source port 80, destination port 11003

(depth/weight/total drops/no-buffer drops/interleaves) 1/256/0/0/0
Conversation 42, linktype: ip, length: 154
source: 155.1.146.6, destination: 155.1.45.5, id: 0x874A, ttl: 254, prot: 1

(depth/weight/total drops/no-buffer drops/interleaves) 5/5397/601/0/0
Conversation 5, linktype: ip, length: 64
source: 155.1.146.6, destination: 155.1.45.5, id: 0x02B8, ttl: 254,
TOS: 184 prot: 17, source port 50700, destination port 16384
```

There are three flows active in the queue. The total queue length is 512, which is what we set the hold-queue to. A total of 12 packets are currently queued up. Note that packet sizes include layer 2 overhead, which is 4 bytes for HDLC.

The first displayed flow corresponds to the HTTP traffic, and has a maximum accumulated queue depth compared to the other flows. Its weight is 256, which corresponds to 25% of the interface bandwidth, or 32Kbps (note that CBWFQ does not take the IP Precedence of the HTTP traffic into account). The ICMP traffic has the same weight, while the flow corresponding to VOICE class has conversation number 5. This means it uses a dynamic queue for scheduling, and its weight of 5397 is actually $32384/(5+1)$ – the automatic weight corresponding to IP Precedence 5 traffic. We can calculate the guaranteed bandwidth shares based on observed weights using the following computations.

First, the bandwidth is shared in values inversely proportional to computation weights:

$1/256:1/256:1/5397$.

Normalize the proportion by dividing by the smallest number:

$5397/256:5397/256:1 = 21:21:1$.

Now the actual bandwidth shares are (roughly):

$[21/(21+21+1), 21/(21+21+1), 1/(21+21+1)] * 100\% = 49\% 49\% 2\%$.

Therefore the net effect is that the available bandwidth is almost equally divided between the two user-defined classes, and the dynamic conversation gets a much smaller share. To confirm this, verify the IP SLA statistics on R6, and note that conversation statistics for the VOICE traffic exhibits the largest number of packet drops.

Rack1R6#show ip sla monitor statistics 1

```
Round trip time (RTT)   Index 1
  Latest RTT: 6 ms
Latest operation start time: 04:09:04.672 UTC Fri Aug 15 2008
Latest operation return code: OK
RTT Values
  Number Of RTT: 29
  RTT Min/Avg/Max: 37/228/434 ms
Latency one-way time milliseconds
  Number of one-way Samples: 0
  Source to Destination one way Min/Avg/Max: 0/0/0 ms
  Destination to Source one way Min/Avg/Max: 0/0/0 ms
Jitter time milliseconds
  Number of Jitter Samples: 27
  Source to Destination Jitter Min/Avg/Max: 5/13/17 ms
  Destination to Source Jitter Min/Avg/Max: 1/1/3 ms
Packet Loss Values
  Loss Source to Destination: 655      Loss Destination to Source: 64
  Out Of Sequence: 0      Tail Drop: 252      Packet Late Arrival: 64
Voice Score Values
  Calculated Planning Impairment Factor (ICPIF): 40
MOS score: 2.75
Number of successes: 136
Number of failures: 18
Operation time to live: Forever
```

Even though the traffic is marked with IP Precedence 5, CBWFQ penalizes the flow as compared to the other user-defined classes. The user-defined flows always get what they ask for, and are always preferred over the dynamic conversations.

10.35 MQC Bandwidth Percent

- Modify R4's previous user-defined reservations to use a percentage reservation of 25% of the link bandwidth each, as compared to 32Kbps.

Configuration

```
R4:
policy-map SERIAL_LINK
  class HTTP
    no bandwidth
    bandwidth percent 25
  class SCAVENGER
    no bandwidth
    bandwidth percent 25
```

Verification

Note

Recall that CBWFQ does not use the absolute **bandwidth** value in a class to directly compute the scheduling weight, but instead it is computed relative to the interface's bandwidth value as $Weight(i) = Const * InterfaceBandwidth / Bandwidth(i)$. Since this value is in reality a relative reservation in the first place, the calculation can be simplified by ignoring the interface level bandwidth, and expressing the reservation as a percentage. In this manner with the **bandwidth percent** command the weight calculation changes to:

$$Weight(i) = Const * 100 / Percent(i)$$

This implementation is useful in designs where a common template of reservation is applied to multiple links of differing bandwidth values. For example a standard policy-map could be defined to reserve 20% of link bandwidth for HTTP flows, and then applied to both 100Mbps FastEthernet and 45Mbps DS3. The resulting CBWFQ weights would be the same, but the actual bandwidth value differs based on the underlying physical link bandwidth.

Like the Kbps reservation through the **bandwidth** command, the sum of all configured **bandwidth percent** values in all classes of a policy-map cannot exceed the **max-reserved-bandwidth** configured on the respective interface. In addition, it is not possible to mix the **bandwidth** and **bandwidth percent** commands in the same policy-map; the units must be the same among classes.

Verification of this configuration is the same as the previous task. Shut down R5's Frame Relay link to force the traffic from VLAN 146 to the hosts behind R4 to take the path across the serial link. R6 should send IP SLA jitter probes to R5 across the serial link; this flow simulates voice traffic and has no explicit bandwidth set under the class. Next, configure R1 as an HTTP server, and set SW2 to transfer the IOS image from R1, oversubscribing the serial link. Lastly, generate a flow of ICMP packets from R6 to R5, simulating the SCAVENGER class traffic.

```
R1:
ip http server
ip http path flash:
```

```
R4:
interface Serial 0/1
 load-interval 30
```

```
R5:
rtr responder
!
interface Serial 0/0
 shutdown
```

```
R6:
ip sla monitor 1
 type jitter dest-ipaddr 155.1.45.5 dest-port 16384 codec g729a
 timeout 1000
 frequency 1
!
ip sla monitor schedule 1 life forever start-time now
```

```
Rack1SW2#copy http://admin:cisco@155.1.146.1/c2600-adventerprisek9-
mz.124-10.bin null:
```

```
Rack1R6#ping 155.1.45.5 repeat 100000 size 150
```

Type escape sequence to abort.

Sending 100000, 150-byte ICMP Echos to 155.1.45.5, timeout is 2 seconds:

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

Check the policy-map statistics as follows.

Rack1R4#show policy-map interface serial 0/1

Serial0/1

Service-policy output: SERIAL_LINK

```
Class-map: VOICE (match-all)
  183205 packets, 11725120 bytes
  30 second offered rate 23000 bps, drop rate 0 bps
  Match: access-group name VOICE
  Match: packet length min 60 max 60
  QoS Set
    dscp ef
    Packets marked 183205

Class-map: HTTP (match-all)
  50438 packets, 29248267 bytes
  30 second offered rate 105000 bps, drop rate 0 bps
  Match: access-group name HTTP
  QoS Set
    precedence 2
    Packets marked 50438
  Queueing
    Output Queue: Conversation 41
    Bandwidth 25 (%)
    Bandwidth 32 (kbps)Max Threshold 64 (packets)
    (pkts matched/bytes matched) 361/208844
    (depth/total drops/no-buffer drops) 2/0/0

Class-map: LARGE_ICMP (match-all)
  0 packets, 0 bytes
  30 second offered rate 0 bps, drop rate 0 bps
  Match: protocol icmp
  Match: packet length min 1001
  drop

Class-map: SCAVENGER (match-all)
  41195 packets, 6331150 bytes
  30 second offered rate 16000 bps, drop rate 0 bps
  Match: input-interface FastEthernet0/1
  Match: ip precedence 0
  QoS Set
    precedence 1
    Packets marked 107619
  Queueing
    Output Queue: Conversation 42
    Bandwidth 25 (%)
    Bandwidth 32 (kbps)Max Threshold 64 (packets)
    (pkts matched/bytes matched) 196/30114
    (depth/total drops/no-buffer drops) 1/0/0

Class-map: class-default (match-any)
  965 packets, 218264 bytes
  30 second offered rate 0 bps, drop rate 17000 bps
  Match: any
  Queueing
    Flow Based Fair Queueing
    Maximum Number of Hashed Queues 32
    (total queued/total drops/no-buffer drops) 64/18239/0
```

This output displays the percent values along with inferred absolute bandwidth reservation, where $\text{bandwidth} = \text{percent} * \text{interface_bandwidth}$. Other than this the bandwidth shares remain the same. Next look at the queue content and note the CBWFQ weights.

Rack1R4#show queueing interface serial 0/1

```
Interface Serial0/1 queueing strategy: fair
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 28259
  Queueing strategy: Class-based queueing
  Output queue: 70/512/32/28259 (size/max total/threshold/drops)
    Conversations 3/6/32 (active/max active/max total)
    Reserved Conversations 3/3 (allocated/max allocated)
    Available Bandwidth 40 kilobits/sec

  (depth/weight/total drops/no-buffer drops/interleaves) 1/256/0/0/0
  Conversation 42, linktype: ip, length: 154
  source: 155.1.146.6, destination: 155.1.45.5, id: 0x9D03, ttl: 254, prot: 1

  (depth/weight/total drops/no-buffer drops/interleaves) 5/256/0/0/0
  Conversation 41, linktype: ip, length: 580
  source: 155.1.146.1, destination: 155.1.58.8, id: 0x98D6, ttl: 254,
  TOS: 64 prot: 6, source port 80, destination port 11003

  (depth/weight/total drops/no-buffer drops/interleaves) 64/5397/686/0/0
  Conversation 27, linktype: ip, length: 64
  source: 155.1.146.6, destination: 155.1.45.5, id: 0x02ED, ttl: 254,
  TOS: 184 prot: 17, source port 51424, destination port 16384
```

Just as before, flows 41 and 42 are assigned weights of 256, while the VOICE flow with a weight of 5397 is heavily penalized as compared to the user-defined classes.

10.36 MQC LLQ and Remaining Bandwidth Reservations

- Modify R4's policy to R5 as follows:
 - Configure Low Latency Queueing to allow for exactly one VoIP call to be prioritized
 - Additional VoIP calls should be allowed only if the link is not congested, but they should not be prioritized
 - Reserve 33% of the remaining bandwidth to the HTTP and SCAVENGER classes each
- Assume that VoIP calls are using the G.729 codec, which generates 50 packets per second with a Layer 3 size of 60 bytes.

Configuration

```
R4:
policy-map SERIAL_LINK
  class VOICE
    priority 27
  class HTTP
    no bandwidth
    bandwidth remaining percent 33
  class SCAVENGER
    no bandwidth
    bandwidth remaining percent 33
```

Verification

Note

The Low Latency Queueing (LLQ) feature is the MQC equivalent of the legacy IP RTP Priority, however classification is not limited to just RTP packets. Like IP RTP Priority, LLQ uses the special conversation number 2^N+8 , where N defines the number of dynamic conversations. For example, if there are 32 (2^5) dynamic queues, then the LLQ conversation is number 40. This conversation has a weight value of 0, which means that it is always serviced first. In order to prevent starvation of other queues, the packets de-queued from the LLQ conversation are metered using a simple token bucket, with a configurable rate and burst size. Packets that exceed the token bucket are dropped (policed) during times of congestion, while if there is no congestion, exceeding traffic is not dropped, but it is simply not prioritized. For these reasons it is better to use the MQC LLQ feature in practical designs versus the legacy priority-queue or IP RTP Priority, because the legacy priority-queue does not have a policer, IP RTP Priority can not classify other traffic flows, and neither of them can be used in conjunction with other QoS features, such as a bandwidth reservation.

Multiple classes inside a single policy-map can use the priority keyword, but only one single priority queue exists. This design of multiple priority classes is used to ensure that one priority flow does not starve another priority flow. For example assume that VoIP and Video traffic are grouped together in one class-map, which has a **priority 128** value defined under a policy-map. If Video traffic is using 128Kbps, VoIP traffic can potentially be dropped (policed), or at least not be guaranteed priority. This is because both traffic flows must contend for the same 128Kbps rate. However if separate VoIP and Video class-maps are defined, each with a **priority 64** rate configured under the same policy-map, each flow is guaranteed priority up to 64Kbps. Although this still totals 128Kbps of priority, one class can't completely starve the other.

Note that the LLQ policer takes the layer 2 packet length into account, so in this case the HDLC overhead of 7 bytes is added to the 60 bytes of layer 3 VoIP payload, which results in a value close to 27Kbps ($67 \text{ bytes/packet} * 50 \text{ packets/second} * 8 \text{ bits/byte} = 26800\text{bps}$) being reserved. The burst value for the token bucket can be configured manually, or automatically calculated by the IOS itself. A burst size of 1 or 1.5 seconds should be enough, since this is probably the maximum duration of a single spoken word in VoIP. See the *Further Learning* section following the verification for more info about computing LLQ sizes for VoIP calls.

Once the LLQ priority reservation is configured, the CBWFQ algorithm subtracts the reserved bandwidth of the priority queue from the interface's available bandwidth. Like in IP RTP Priority, the available bandwidth value defaults to 75% of the interface bandwidth, and can be modified with the **max-reserved-bandwidth** interface level command. The remaining bandwidth can be used to create a relative bandwidth reservation for other classes in the CBWFQ. By this logic, instead of configuring absolute bandwidth values - either numerically or in interface bandwidth percents - for other reservations, you can just specify relative shares of the bandwidth that remains after the priority queue finished its run. This method is seen in the above configuration through the **bandwidth remaining percent** command.

For example in this particular task the link between R4 and R5 was configured with a interface **bandwidth** value of 128Kbps, and a **max-reserved-bandwidth** of 100, meaning that 100% of 128Kbps is reservable. With the 27Kbps reservation for VoIP in the LLQ, 101Kbps is the remaining bandwidth. Since the HTTP and SCAVENGER classes reserve 33% of the remaining bandwidth, they are each allocated a minimum of 33Kbps ($(128-27) * .33 = 33$).

To verify this configuration generate three different traffic flows, like in the previous examples. First, shutdown the Frame Relay interface of R5 to force the traffic from VLAN 146 to hosts behind R5 to take path across the serial link. Configure R6 to send IP SLA jitter probes to R5; this flow simulates the voice traffic in the LLQ. Next, configure R1 as an HTTP server, and set SW2 to transfer the IOS image from R1, causing the serial link to be congested. Lastly, generate a flow of ICMP packets from R6 to R5, simulating the SCAVENGER class traffic.

```
R1:
ip http server
ip http path flash:
```

```
R4:
interface Serial 0/1
 load-interval 30
```

```
R5:
rtr responder
!
interface Serial 0/0
 shutdown
```

```
R6:
ip sla monitor 1
 type jitter dest-ipaddr 155.1.45.5 dest-port 16384 codec g729a
 timeout 1000
 frequency 1
!
ip sla monitor schedule 1 life forever start-time now
```

```
Rack1SW2#copy http://admin:cisco@155.1.146.1/c2600-adventerprisek9-
mz.124-10.bin null:
```

```
Rack1R6#ping 155.1.45.5 repeat 100000 size 150
```

Type escape sequence to abort.

Sending 100000, 150-byte ICMP Echos to 155.1.45.5, timeout is 2 seconds:

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

Check the statistics for policy map matches. Note that the priority queue for VOICE traffic uses conversation 40, and its 30 seconds offered rate is 25Kbps. Since the voice packets use the priority queue now, the remaining classes have less bandwidth to share. Therefore the bandwidth measured for the HTTP class is only 78Kbps, compared to the higher value available seen in prior configurations without the LLQ. Also, note that SCAVENGER class does not use all its guaranteed bandwidth, so the HTTP class can use the remaining part.

```
Rack1R4#show policy-map interface serial 0/1
Serial0/1
```

```
Service-policy output: SERIAL_LINK
```

```
Class-map: VOICE (match-all)
 26349 packets, 1686336 bytes
 30 second offered rate 25000 bps, drop rate 0 bps
Match: access-group name VOICE
Match: packet length min 60 max 60
QoS Set
  dscp ef
  Packets marked 26349
Queueing
  Strict Priority
  Output Queue: Conversation 40
  Bandwidth 27 (kbps) Burst 3400 (Bytes)
  (pkts matched/bytes matched) 26274/1681536
  (total drops/bytes drops) 0/0

Class-map: HTTP (match-all)
 9491 packets, 5504156 bytes
 30 second offered rate 78000 bps, drop rate 0 bps
Match: access-group name HTTP
QoS Set
  precedence 2
  Packets marked 9491
Queueing
  Output Queue: Conversation 41
  Bandwidth remaining 33 (%)Max Threshold 64 (packets)
  (pkts matched/bytes matched) 3092/1793048
  (depth/total drops/no-buffer drops) 6/0/0

Class-map: LARGE_ICMP (match-all)
 0 packets, 0 bytes
 30 second offered rate 0 bps, drop rate 0 bps
Match: protocol icmp
Match: packet length min 1001
drop
```

```

Class-map: SCAVENGER (match-all)
 8739 packets, 1343986 bytes
 30 second offered rate 19000 bps, drop rate 0 bps
Match: input-interface FastEthernet0/1
Match: ip precedence 0
QoS Set
  precedence 1
    Packets marked 75162
Queueing
  Output Queue: Conversation 42
  Bandwidth remaining 33 (%)Max Threshold 64 (packets)
  (pkts matched/bytes matched) 2843/437262
  (depth/total drops/no-buffer drops) 1/0/0

Class-map: class-default (match-any)
 139 packets, 31440 bytes
 30 second offered rate 0 bps, drop rate 0 bps
Match: any
Queueing
  Flow Based Fair Queueing
  Maximum Number of Hashed Queues 32
  (total queued/total drops/no-buffer drops) 0/0/0

```

View the CBWFQ contents, and note that the flow corresponding to the VOICE class has weight value of zero, and CBWFQ no longer penalizes it by excessive packet drops. Also, note the weight of two other remaining flows (HTTP and SCAVENGER classes). The value is "194", which is actually 64/0.33. Thus both flows have equal chances of claiming the bandwidth remaining after the priority queue.

Rack1R4#show queueing interface serial 0/1

```

Interface Serial0/1 queueing strategy: fair
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 29854
  Queueing strategy: Class-based queueing
  Output queue: 8/512/32/29854 (size/max total/threshold/drops)
    Conversations 3/6/32 (active/max active/max total)
    Reserved Conversations 3/3 (allocated/max allocated)
    Available Bandwidth 101 kilobits/sec

  (depth/weight/total drops/no-buffer drops/interleaves) 1/0/0/0/0
  Conversation 40, linktype: ip, length: 64
  source: 155.1.146.6, destination: 155.1.45.5, id: 0x02A1, ttl: 254,
  TOS: 184 prot: 17, source port 51032, destination port 16384

  (depth/weight/total drops/no-buffer drops/interleaves) 1/194/0/0/0
  Conversation 42, linktype: ip, length: 154
  source: 155.1.146.6, destination: 155.1.45.5, id: 0xA6B0, ttl: 254, prot: 1

  (depth/weight/total drops/no-buffer drops/interleaves) 6/194/0/0/0
  Conversation 41, linktype: ip, length: 580
  source: 155.1.146.1, destination: 155.1.58.8, id: 0xA432, ttl: 254,
  TOS: 64 prot: 6, source port 80, destination port 11003

```



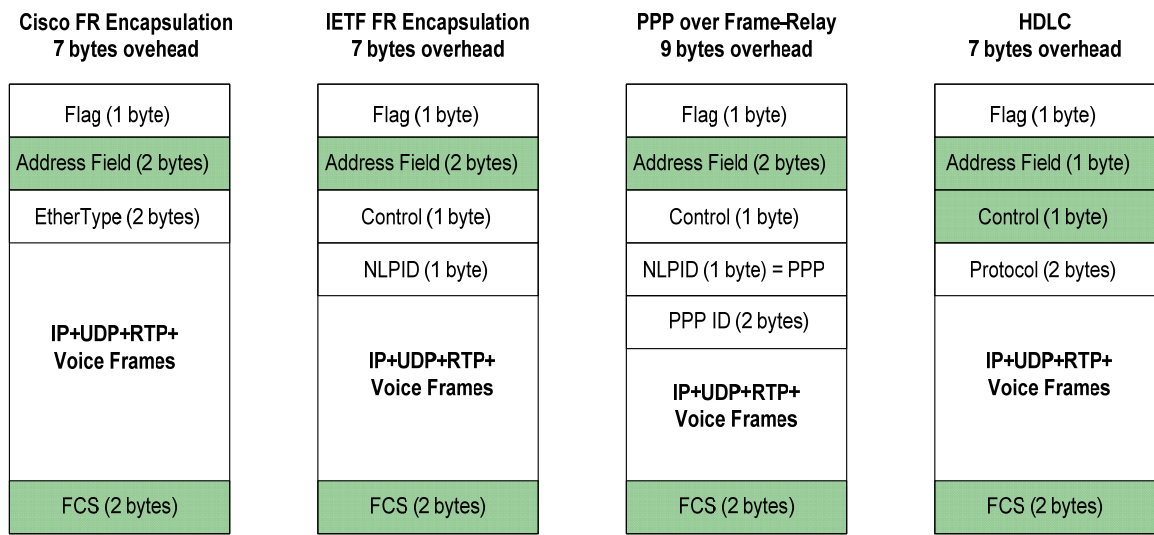
Further Learning

Computing voice bandwidth is usually required for scenarios where you provision an LLQ based on the number of calls and VoIP codec used. To accomplish this you need to account for codec payload rate, the Layer 3 overhead (RTP and UDP headers), and the Layer 2 overhead (Frame-Relay, Ethernet, HDLC etc. headers). Accounting for Layer 2 overhead is important since the LLQ policer takes this overhead in account when enforcing the maximum rate.

For this example let's consider two codecs for bandwidth computation, G.729 and G.711. By default both codecs generate 50 VoIP packets per second, however the codec framing rate is 10ms (100 packets per second). Thus each VoIP packet carries two frames with VoIP samples. The frame sizes are 10 bytes and 80 bytes for G.729 and G.711 respectively. Therefore G.729 generates $(10 \times 2) \times 50 \times 8 = 8000\text{bps}$ and G.711 generates $(80 \times 2) \times 50 \times 8 = 64000\text{bps}$ of payload rate.

The RTP header size is 12 bytes, and the UDP header size is 8 bytes. A typical IP header (with no options) is 20 bytes. Therefore, the Layer 3 overhead is 40 bytes, if we don't use header compression.

The following are the formats WAN frames commonly used to transport voice (with or without FRF.12/MLP fragmentation – voice packets are never fragmented).



As we can see both Cisco and IETF Frame-Relay encapsulations add 7 bytes of Layer 2 overhead to VoIP packets. The same holds true for HDLC encapsulation (which is not very common but added here for sake of completeness). PPP over Frame-Relay adds 9 bytes of overhead – the maximum overhead of all encapsulation types.

Using the information above, you can compute bandwidth usage for uncompressed voice traffic flow across any WAN connection. For example, let's compute the bandwidth consumption for 3 G.729 calls across Frame-Relay link with FRF.12 fragmentation. First off, FRF.12 does not fragment voice packets if set up properly, as the fragment size is greater than or equal to the VoIP packet. Thus we assume 7 bytes of Layer 2 overhead for any of the Frame Relay encapsulation types. Next, the size of the payload + Layer 3 overhead is 20 bytes + 40 bytes = 60 bytes. Based on the 50pps rate we end up with the bandwidth value of $(20+40+7)*50*8=26800\text{bps}$. If you want to use the G.711 codec, then replace the 20 bytes payload with 160 bytes. The result is $(160+40+7)*50*8=82800\text{bps}$.

Another thing to consider is IP/RTP/UDP header compression. Cisco's implementation reduces the total overhead of 40 bytes (12+8+20) down to 2 bytes (no UDP checksum). This limits the Layer 3 overhead to just 2 bytes. Based on this reduction from 40 to 2 bytes we could compute the bandwidth usage for a G.729 call over MLPoFR with UDP header compression as $(20+2+9)*50*8=12400\text{bps}$. The same computations for compressed G.729 over Frame Relay with or without FRF.12 yields $(20+2+7)*50*8=11600\text{bps}$.

For VoIP over Ethernet the Layer 2 overhead for is typically 18 bytes – 14 bytes for the Ethernet header and 4 bytes for FCS (32 bits). If the frame carries a VLAN tag, add another 4 bytes, for 22 bytes of total overhead. Note that you typically see the G.711 codec used over LAN links.