QoS Solutions

10.1 Hold-Queue and Tx-Ring

- Configure R1's connection to VLAN 146 to have an input software queue length of 10 packets, and an output software queue length of 30 packets.
- Set the output hardware queue size to 15 packets.

Configuration

```
R1:
interface FastEthernet0/0
 tx-ring-limit 15
 hold-queue 10 in
 hold-queue 30 out
```

Verification

```
Rack1R1#show interfaces fastEthernet 0/0 | inclue queue
  Input queue: 0/10/0/0 (size/max/drops/flushes); Total output drops: 0
  Output queue: 0/30 (size/max)

Rack1R1#show controllers fastEthernet 0/0 | include tx
 rx ring entries=64, tx ring entries=128
 txring=0x7DC5800, txr shadow=0x84C0A6D4, tx_head=12, tx_tail=12, tx_count=0
 tx_one_col_err=0, tx_more_col_err=0, tx_no_enp=0, tx_deferred_err=0
 tx_underrun_err=0, tx_late_collision_err=0, tx_loss_carrier_err=6
 tx_exc_collision_err=0, tx_buff_err=0, fatal_tx_err=0
 tx_limited=0(15)
```

> **Note**
>
> Network interfaces on a router work asynchronously. This means the interface driver responds to hardware interrupts, notifies the switching process, and queues incoming packets if the central processor is not yet ready to start working on them. The same applies to outgoing packets. The IOS prepares the packet, puts it into a buffer, and notifies the interface driver of the new packet.
>
> On the input direction if the processor is not too busy and the packet rate is not too high, the system will never actually use the input queue. Similarly, if the outgoing interface is not congested, the interface driver will work fast enough to stop the outgoing queue from growing.

To fully understand an end-to-end QoS model it's important to understand the key differences and similarities between the input and output queues. Both queue types consume memory chunks from buffers pools – interface buffer pools and public shared buffer pools. The buffer management scheme can be quite complicated, especially with particle buffer pools, but the idea remains the same. Queues consume I/O memory available via system buffers, which is a configurable parameter.

For input queueing there is just one queue per interface, which is always First-In-First-Out (FIFO), and has a size of 75 packets by default. The Interface driver assigns incoming packets directly into the incoming FIFO queue if the central processor has no time to switch them quickly (e.g. if sudden packet burst occurs). For more information about input queue management see the section dedicated to SPD in this document.

For output queueing there are two output queues by default. This value can be increased for interfaces that support Virtual Circuits (VCs), such as Frame Relay or ATM, when per-VC queuing is enabled. The first output queue is the "software queue", and defines the per-interface logical queuing strategy. This software queue can be a simple FIFO (hold-queue), or where more advanced "fancy queueing" methods are defined, such as Weighted Fair Queueing (WFQ) or Class-Based Weighted Fair Queueing (CBWFQ). Note that the FIFO hold-queue is the default and simplest queuing strategy on high-speed interfaces.

The second output queue sits right after the software queue, and is the hardware queue or "transmit ring" (TX-ring). The interface driver actually works with this memory space directly when looking for packets to send. This queue is typically smaller than the software queue, and is always FIFO.

Note that the software output queue only starts to fill up when the tx-ring is full. This is due to the fact that traffic does not need to be software queued unless the physical hardware queue is busy sending a packet. Therefore the tx-ring length defines how fast the system switches to a logical queuing strategy for interface scheduling. Triggering complicated software queues such as CBWFQ too often may degrade the overall router performance, such as when CBWFQ needs to make a full sorting run across all queues. This is why tuning the tx-ring and output queue size can be a complicated practice, especially when working with highly delay and jitter sensitive traffic such as voice.

Note that for the software output queue is most commonly referred to as just the "output queue", while the hardware output queue is referred to as just the "transmit ring" (tx-ring). For the sake of differentiating the two these naming conventions are adhered to throughout the scope of this document.

To test the above configuration simulate a packet storm towards R1 and see how the input queue starts to fill up.

```
Rack1R6#ping 155.1.146.1 repeat 1000000 timeout 0 size 1000

Type escape sequence to abort.
Sending 1000000, 1000-byte ICMP Echos to 155.1.146.1, timeout is 0
seconds:
.............................................................
.....................................................!.......
.............................................................
.............................................................
<snip>

Rack1R1#show interfaces fastEthernet 0/0 | inc queue
  Input queue: 11/10/133/0 (size/max/drops/flushes); Total output drops: 0
  Output queue: 0/30 (size/max)
```


10.2 Weighted Fair Queuing (WFQ)

- Configure the point-to-point Serial link between R4 and R5 with an interface clock rate and interface bandwidth of 128Kbps.
- Set the output hold-queue size to 256.
- Configure WFQ on the links with a length of 16 for the congestive discard threshold, 128 for the number of conversations, and 8 for RSVP reservable queues.
- Set the tx-ring size to the minimal value to engage the WFQ as fast as possible.
- Normalize the packet flows for the link by ajdusting the MTU so that each IP packet takes no more than 10ms to be sent.

Configuration

```
R4:
interface Serial0/1
 clock rate 128000
 bandwidth 128
 tx-ring-limit 1
 fair-queue 16 128 8
 hold-queue 256 out
 ip mtu 156

R5:
interface Serial0/1
clock rate 128000
 bandwidth 128
 tx-ring-limit 1
 fair-queue 16 128 8
 hold-queue 256 out
 ip mtu 156
```

Verification

Note

WFQ uses an intelligent congestion management solution that provides “fair” sharing of the interface bandwidth between multiple traffic *flows*. A traffic “flow” (or *conversation*) is a unidirectional sequence of packets, defined based on the protocol type, the source/destination IP addresses, the source/destination ports numbers (when available), and partially on the IPv4 ToS byte value. For example, an HTTP file transfer between two hosts represents one packet flow, while ICMP packets sent from one host to another represents a second.

The term “fair” on WFQ refers to the *max-min* fairness. The WFQ calculation procedure is as follows.

First, divide the interface bandwidth equally between all flows. For example if there are 10 flows, and 100Kbps of bandwidth, each flow gets 10Kbps. If a flow demand is less than the “equal” share, e.g. a flow only needs 5Kbps instead of 10Kbps, allocate the unneeded bandwidth equally among the remaining flows. If there are flows that demand more than the equal share, e.g. the equal share is 10Kbps, but two flows demand 25Kbps and 20Kbps respectively, they will each get the equal, maximum, possible shares (e.g 19Kbps and 19Kbps) but only if there are flows which demand less than the equal share.

In this context *max-min* means that all flows first get the bare minimum, but the procedure tries to maximize each flow’s share if possible. The concept of basic WFQ is very important to understand as many other congestion management techniques utilize it, such as Round Robin scheduling. To reiterate, the key fact in WFQ is that the *max-min* scheduling allows the sharing of a flow’s unclaimed bandwidth between other flows.

An individual flow in the queue may be more or less demanding than other flows. The “demanding” flow generates the higher bit-rate, either due to larger packet sizes or a higher packet per second rate. For example compare a bulk FTP file transfer against telnet sessions or VoIP RTP streams. Within the context of flow-based sharing, it is also important to understand that a single host may generate *multiple* flows, such as with with P2P file-sharing applications or download “accelerators”, thus this particular host can obtain an “unfair” share of bandwidth compared to other hosts. This, unfortunately, is a limitation of a classification scheme that based on simple flows, such as WFQ, which has no notion of “flow mask”.

To enhance its scheduling logic, WFQ may assign a *weight* value to a flow. The weight affects the minimum guaranteed share of bandwidth available to a flow. If there are *N* flows with weights w1, w2 ... w*N*, then flow *K* will get the minimum share of bandwidth where s*K*=(w1+w2+… w*K*+… w*N*)/w*K* – inversely proportional to its weight. The shares are relative, in the sense that the scheduler divides the available bandwidth in proportions: s1:s2:…s*N*.

IOS implementation of WFQ assigns weights automatically based on the IP Precedence (IPP) value in the packet’s IP header. The formula is Weight=32384/(IPP+1), where IPP is the IP Precedence of the flow.

To understand the effect of the weight settings, imagine four flows with different IP Precedence values of zero, one, one and three.

Step 1:
Using the above formula for weight, we obtain:

Weight(1) = 32384/(0+1) = 32384
Weight(2) = 32384/(1+1) = 16192
Weight(3) = 32384/(1+1) = 16192
Weight(4) = 32384/(3+1) = 8096

Step 2:

Compute the sum of all weights:
Sum(Weight(i),1…4) = 32384+16192*2+8096 = 72864

Step 3:
Compute the shares:

Share(1) = 72864/Weight(1) = 72864/32384 = 2.25
Share(2) = 72864/Weight(2) = 72864/16192 = 4.5
Share(3) = 72864/Weight(3) = 72864/16192 = 4.5
Share(4) = 72864/Weight(4) = 72864/8096 = 9

The proportion is 2.25:4.5:4.5:9 = 1:2:2:4 – this is based on the "shifted" ip precedences: (IPP(i) + 1)

Note that the numerator "32384" is arbitrary to those computations, meaning that you can use any value. What is important however is the proportion of "shifted" IP precedences.

For bandwidth sharing to be correct, all flows must be adaptive. Adaptive means that a flow must respond to packet drops by slowing its sending rate. Non-adaptive, aggressive flows may defeat the bandwidth sharing logic of WFQ by claiming all available *buffer space* and starving other flows. This is directly linked to the fact that all flows use shared buffer space when implementing WFQ (more on this later).

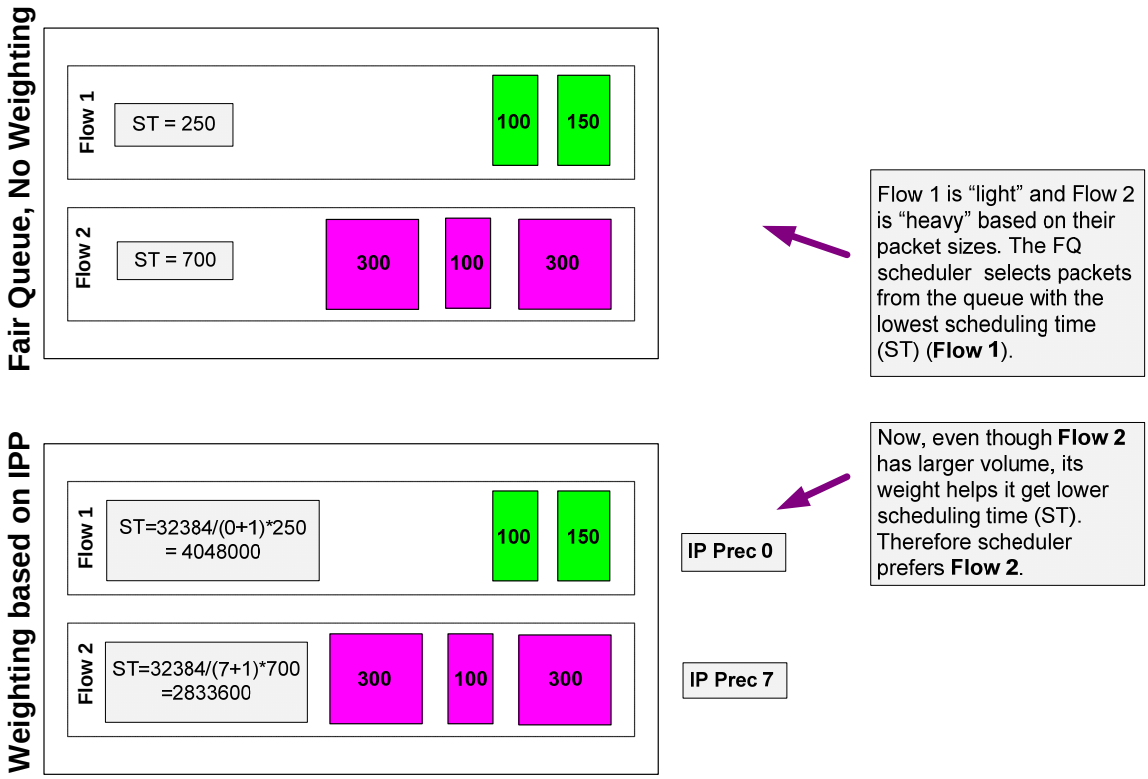
Specifically WFQ implements its fair sharing logic as follows. First, the scheduler creates a group of flow queues for the interface, e.g. 256 "conversations", based on a manual setting or an automatic calculation derived from the interface bandwidth. When a new packet arrives for output scheduling, the scheduler applies a special *hash* function to the packet's source/destination IP/Port values to yield the flow queue number. This is why the number of queues is always a power of 2, because the hash the output value is somewhere between 0 and 2^N. This procedure also means that multiple flows may share the same queue, called "hash collision", when the number of flows is large.

Each flow queue has a special *virtual scheduling time* assigned – the amount of "time" that would take to serialize the packets in the queue across the output interface. This "virtual time" is actually the total size of all packets stored in the flow queue scaled by the flow computational weight.

Now imagine a new packet of size *packet_size* and IP precedence *packet_ip_precedence* arrives to its respective flow queue (hash queue):

weight = 32384/(packet_ip_precedence +1)
*virtual_scheduling_time = queue_tail_time + packet_size*weight*
queue_tail_time = virtual_scheduling_time

The "*queue_tail_time"* variable stores the previous virtual scheduling time for the flow. Note that the weight is inversely proportional to IP precedence, and thus more important packets have smaller virtual scheduling time value (WFQ thinks that it can serialize them "faster"). It is more appropriate to call this computational weight the "scaling factor" to avoid confusion with the classic meaning of weight.



Every time WFQ needs to take a packet out of all queues, it sorts all queues based on their virtual scheduling time (sorting is time-consuming). The scheduler selects the flow queue with *minimum* virtual scheduling time, sends a packet out and then decreases the respective *queue_tail_time* variable by packet's size. In essence, this procedure tries to pull the short (smaller packets) flows traffic first.

However, the following question remains open: if a new flow queue has just started, how should we initialize its *queue_tail_time* variable? Should we do that based on just the first packets size? That would result in every new flow having better chances of being scheduled then the older ones, with larger *queue_tail_times*! This is why WFQ stores a special variable called *round_number* shared by all queues. The *round_number* is the *virtual_scheduling_time* of the last packet sent down the physical line. When a flow queue is first initialized, its *queue_tail_time* is set equal to the latest *round_number,* thus making it "comparable" with the existing flow queues, and not letting it monopolize the physical line. The idea is to penalize the new flow by the amount of time other flows have already waited so far.

The next important point is the congestive discard threshold (CDT), which is a unique congestion avoidance scheme available with WFQ. First you configure the total buffer space allocated to WFQ using the **hold-queue <*N*> out** command. The WFQ shares this buffer space between all flow queues. Any queue may grow up to the maximum free space, but as soon as its size reaches the CDT, WFQ drops a packet from a flow queue with the *maximum* virtual scheduling time (this may be some other queue, not the one that crossed the packet threshold). This way, WFQ penalizes the most aggressive flow and triggers a mechanism similar to random early detection's (RED) prevention of tail-drop. The fair-queue settings command is **fair-queue <*CDT*> <*N Flow Queues*> <*N Reservable Queues*>.**

The number of reservable conversations (queues) is the number of flow-queues available to RSVP reservations (if any). Those flows have a very small weight value, and thus are preferred above any other flows. In addition to reserved flow queues, there are special "Link Queues". The number of queues is fixed to 8, and they are numbered right after the maximum dynamic queue (e.g. if there are 32 dynamic queues, "Link Queues" start at number 33). WFQ uses those queues to service routing protocol traffic and Layer 2 keepalives – everything that is critical to router operations and management. Each queue has a weight 1024, which is lower than any dynamic queue can get, so control plane traffic has priority over regular data traffic.

Finally, note that the interface bandwidth setting does not influence the WFQ algorithm directly. It only prompts the WFQ command to pick up optimal parameters matching the interface bandwidth. The bandwidth, however, *is* used for admission control with RSVP flows.

For this particular task, verification can be performed as follows.

```
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: 2044
  Queueing strategy: weighted fair
  Output queue: 3/256/16/2044 (size/max total/threshold/drops)
     Conversations  2/3/32 (active/max active/max total)
     Reserved Conversations 0/0 (allocated/max allocated)
     Available Bandwidth 96 kilobits/sec

Rack1R4#show queueing fair
Current fair queue configuration:

  Interface           Discard    Dynamic  Reserved  Link     Priority
                      threshold  queues   queues    queues   queues
  Serial0/0           64         256      0         8        1
  Serial0/1           64         32       1         8        1
```

Changing the **ip mtu** for the interface in this task causes fragmentation at layer 3, and normalizes the packet sizes for further testing. With an additional 4 bytes of overhead for the layer 2 HDLC encapsulation, every frame sent out the link has a maximum size of 160 bytes (156 of IP plus 4 bytes of HDLC) . With a physical clocking rate of 128000 bits per second, this means a 160 byte packet will take a maximum time of 10ms to serialize. Adjusting serialization delay through fragmentation is covered in more detail later in this document.

To verify how WFQ shares the interface bandwidth, configure the network as follows:

1) Enable HTTP servers on R6 and R1 and set the root directory to “flash:”.

2) Shutdown the Frame-Relay interface of R5 to ensure traffic from VLAN146 takes the path across the serial link between R4 and R5.

3) Configure R6 to mark traffic leaving the VLAN146 interface with an IP precedence of 1, and configure R1 to mark traffic leaving VLAN146 with an IP precedence of 3.

4) Create a policy-map on R5 to match incoming IP precedence 1 and IP precedence 3 packets, and apply it inbound to interface Serial 0/1. We will use this policy to meter incoming traffic.

5) Copy IOS images stored in the flash memory of R1 and R6 to SW2 and SW4’s “null:” destinations using HTTP.

The resulting configuration is as follows.


```
R1:
ip http server
ip http path flash:
!
policy-map MARK
 class class-default
  set ip precedence 3
!
interface FastEthernet 0/0
 service-policy output MARK

R5:
class-map match-all IP_PREC3
  match ip precedence 3
 class-map match-all IP_PREC1
  match ip precedence 1
!
 policy-map METER
  class IP_PREC1
  class IP_PREC3
!
interface Serial 0/1
 service-policy METER input
 load-interval 30
 clock rate 128000
!
interface Serial 0/0
 shutdown

R6:
ip http server
ip http path flash:
!
policy-map MARK
 class class-default
  set ip precedence 1
!
interface FastEthernet 0/0.146
 service-policy output MARK
```

Rack1SW2#copy http://admin:cisco@155.1.146.6/c2600-adventerprisek9-mz.124-10.bin null:

Rack1SW4#copy http://admin:cisco@155.1.146.1/c2600-adventerprisek9-mz.124-10.bin null:

The configuration effectively generates two TCP flows across R4's connection to R5. Since TCP is adaptive, the flows should eventually balance and start sharing bandwidth using their weights: 1+1 and 3+1, which is 2:4. This means file transfers from R1 to SW4 should have twice as much bandwidth as the file transfer from R6 to SW2.

```
Rack1R5#show policy-map int serial 0/1
 Serial0/1

  Service-policy input: METER

    Class-map: IP_PREC1 (match-all)
      91618 packets, 17299811 bytes
      30 second offered rate 41000 bps
      Match: ip precedence 1

    Class-map: IP_PREC3 (match-all)
      330231 packets, 75523458 bytes
      30 second offered rate 83000 bps
      Match: ip precedence 3

    Class-map: class-default (match-any)
      58353 packets, 8839968 bytes
      30 second offered rate 0 bps, drop rate 0 bps
      Match: any
```

Check the flow queues on R4 to see their WFQ parameters. Note that the average packet length is the same, and the inverse weight proportion is 1/8096:1/16192 = 2:1 (the guaranteed shares of bandwidth for IP Precedence 1 and IP Precedence 0 traffic).

```
Rack1R4#show queueing interface serial 0/1
<snip>
  (depth/weight/total drops/no-buffer drops/interleaves) 7/16192/0/0/0
  Conversation 20, linktype: ip, length: 580
  source: 155.1.146.6, destination: 155.1.58.8, id: 0x6927, ttl: 254,
  TOS: 32 prot: 6, source port 80, destination port 11009

  (depth/weight/total drops/no-buffer drops/interleaves) 6/8096/0/0/0
  Conversation 26, linktype: ip, length: 580
  source: 155.1.146.1, destination: 155.1.108.10, id: 0x7BCF, ttl: 254,
  TOS: 96 prot: 6, source port 80, destination port 11001
```


As mentioned above, non-adaptive flows, such as UDP/ICMP traffic floods, may oversubscribe the shared WFQ buffer space. This is due to their "greedy" behavior - a single flow tries to monopolize the whole buffers space available to all queues, causing excessive packet drops, and dose not respond to congestive discards.

To see how this behavior can manifest itself originate two ICMP packet floods from R6 and R1 towards R5 using a packet size of 100 and a timeout of 0. The timeout value of zero means that the devices do not wait for an echo-reply before sending its next echo. Additionally configure R5 so that it does not respond to the echos so the return traffic doesn't affect the flow.

```
R5:
access-list 100 deny icmp any host 155.1.45.5
access-list 100 permit ip any any
!
interface Serial 0/1
 ip access-group 100 in
 no ip unreachables
```

```
Rack1R1#ping 155.1.45.5 repeat 100000000 timeout 0
Rack1R6#ping 155.1.45.5 repeat 100000000 timeout 0
```

```
Rack1R4#show queueing interface serial 0/1
<snip>
  (depth/weight/total drops/no-buffer drops/interleaves) 43/8096/7289/0/0
  Conversation 30, linktype: ip, length: 104
  source: 155.1.146.1, destination: 155.1.45.5, id: 0x05E1, ttl: 254, prot: 1

  (depth/weight/total drops/no-buffer drops/interleaves) 21/16192/25458/0/0
  Conversation 3, linktype: ip, length: 104
  source: 155.1.146.6, destination: 155.1.45.5, id: 0xC38A, ttl: 254, prot: 1
```

```
Rack1R5#show policy-map interface serial 0/1
 Serial0/1

  Service-policy input: METER

    Class-map: IP_PREC1 (match-all)
      104422 packets, 22233791 bytes
      30 second offered rate 47000 bps
      Match: ip precedence 1

    Class-map: IP_PREC3 (match-all)
      353680 packets, 85227338 bytes
      30 second offered rate 68000 bps
      Match: ip precedence 3
<snip>
```

Note the huge number of drops due to the queue getting full. Also, note that offered rates on R5 are *not* in a 1:2 proportion, even though the weights are set to share the bandwidth in 1:2 ratio and the packet lengths are the same.

10.3 Legacy RTP Reserved Queue

- Reset R4's Serial link to R5 to the default queueing method.
- Configure a single command on R4's interface so that 100% of the link bandwidth is reserved for RTP traffic in the UDP port range 16384 – 32767.

Configuration

```
R4:
interface Serial 0/1
 no hold-queue out
 fair-queue
 max-reserved-bandwidth 100
 ip rtp reserve 16384 16383 128
```

Verification

Note

Real Time Protocol (RTP) is the UDP based protocol used for audio and video transmissions, such as VoIP. **`ip rtp reserve`** was the first feature aimed at making WFQ aware of voice bearer traffic (RTP packets). By default, WFQ already prefers voice packets due to their small size and (usually) high IP Precedence value of 5. However as competing flows with the same weight appear in the WFQ, voice traffic is no longer given preferrential treatment.

The first solution for this problem was to enable automatic mapping of voice bearer traffic (RTP packets with a UDP port range 16384-32767) to a special flow queue. WFQ uses a Resource Reservation Protocol (RSVP) classifier and special RSVP conversation to queue the matching packets. The WFQ scheduler will treat this queue as a special "RSVP" reserved queue with a very low weight. Thus, only RSVP flows may compete with the special flow for the voice bearer packets. More details on RSVP and WFQ interaction are covered later.

The command format is **`ip rtp reserve <Starting UDP Port> <Port Range> <Bandwidth>`**

For example **`ip rtp reserve 16384 16383 128`** means to reserve a special RSVP conversation for UDP traffic flows destined to ports in range 16384 – 32767 (16384+16383) – the standard RTP port range used by Cisco devices. The conversation number for IP RTP Reserve is equal to number_of_dynamic_WFQ_flows + 8 + 1, e.g. the first flow after the "Link Queues".

Configuring IP RTP Reserve reduces the "available interface bandwidth" counter by the amount of bandwidth reserved. This counter is needed for the purpose of admission control with RSVP flows, and user-defined classes in CBWFQ. By default, the amount of "available" bandwidth is 75% of total interface bandwidth. The idea is to "underestimate" the bandwidth, allowing some space for routing updates, Layer 2 keepalives, and any other data such as physical layer bit stuffing. This implies that the <*bandwidth*> value cannot be more than <*max-reserved-bandwidth*>*<*interface bandwidth*> - that is, the interface "available" bandwidth. In this task the **`max-reserved-bandwidth 100`** command is used to allow 128Kbps to be reserved.

The WFQ weight assigned to the special reserved queue is a fixed value of 128, and does not depend on the <*bandwidth*> setting. This value is relatively low to be able to preempt any data traffic flow with an IP precedence in the range of 0 through 7, since the best weight value for IP Precedence 7 traffic is 32384/(7+1) = 4048.

However, a special internal policer, with the rate equal to the configured bandwidth, applies to the matched traffic, and all exceeding packets will have a WFQ weight of 32384. This means that traffic in the RTP port range that exceeds the configured bandwidth is treated as if it had an IP Precedence of 0, which equates to best-effort forwarding. This is a special measure to make sure that the reserved conversation does not consume all of the interface bandwidth by using its low weight. Note that exceeding packets are not dropped, just handled by the WFQ scheduler as if they have less important weight.

```
Rack1R4#show queueing fair
Current fair queue configuration:

  Interface           Discard    Dynamic  Reserved  Link     Priority
                      threshold  queues   queues    queues   queues
  Serial0/0           64         256      0         8        1
  Serial0/1           64         32       1         8        1
```


To verify IP RTP reserve in action we will configure R6 to source G.729-like packet streams to R5 through the IP SLA feature, and measure jitter and round-trip time (RTT). R5 will be configured to respond to these probes. At the same time R1 will be configured as an HTTP server and allow other routers to download its IOS image, which will cause additional congestion in the output queue of R4. R6 will mark outgoing packets with an IP Precedence value of 1, and R1 will mark outgoing packets with an IP precedence value of 7 (the maximum priority and minimum WFQ weight). Note that the Frame-Relay interface of R5 is shutdown to make sure R1 prefers to reach R5 through R4, and vice versa.

```
R1:
ip http server
ip http path flash:
!
policy-map MARK
 class class-default
  set ip precedence 7
!
interface FastEthernet 0/0
 service-policy output MARK

R5:
interface Serial 0/0
 shutdown
!
rtr responder

R6:
policy-map MARK
 class class-default
  set ip precedence 1
!
interface FastEthernet 0/0.146
 service-policy output MARK
!
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
```


Disable RTP reserve on R4 and start transferring the IOS image file from R1 to SW4.

```
R4:
interface Serial 0/1
 no ip rtp reserve
```

```
Rack1SW4#copy http://admin:cisco@155.1.146.1/c2600-adventerprisek9-
mz.124-10.bin null:
```

Note the WWW file transfer has a minimum possible weight for a dynamic conversation (4048) based on its high IP precedence value of 7.

```
Rack1R4#show queueing interface serial 0/1
<snip>
  (depth/weight/total drops/no-buffer drops/interleaves) 19/4048/170/0/0
  Conversation 7, linktype: ip, length: 160
  source: 155.1.146.1, destination: 155.1.108.10, id: 0x3E96, ttl: 254, prot: 6

  (depth/weight/total drops/no-buffer drops/interleaves) 1/16192/22/0/0
  Conversation 2, linktype: ip, length: 64
  source: 155.1.146.6, destination: 155.1.45.5, id: 0x000F, ttl: 254,
  TOS: 32 prot: 17, source port 59144, destination port 16384
```

Note the RTT values for IP SLA probes and low mean opinion score (MOS) indicates poor voice quality.

```
Rack1R6#show ip sla monitor statistics 1
Round trip time (RTT)      Index 1
       Latest RTT: 458 ms
Latest operation start time: 07:34:06.306 UTC Mon Aug 4 2008
Latest operation return code: OK
RTT Values
       Number Of RTT: 978
       RTT Min/Avg/Max: 26/468/701 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: 955
       Source to Destination Jitter Min/Avg/Max: 1/5/24 ms
       Destination to Source Jitter Min/Avg/Max: 1/1/11 ms
Packet Loss Values
       Loss Source to Destination: 22          Loss Destination to Source: 0
       Out Of Sequence: 0  Tail Drop: 0 Packet Late Arrival: 0
Voice Score Values
       Calculated Planning Impairment Factor (ICPIF): 32
MOS score: 3.15
Number of successes: 81
Number of failures: 0
Operation time to live: Forever
```


With the current design the voice packets with IP precedence 1 cannot compete with the larger TCP packets that have an IP precedence of 7. This is due to the large mismatch in their weights. Normally VoIP packets would have an IP precedence of 5, which would help somewhat, but IP precedence 7 would still overtake it as a whole. Note the weight values and the average packet sizes.

Now re-enable the RTP reserve feature and compare the weights for both flows once more.

```
R4:
interface Serial 0/1
 ip rtp reserve 16384 16383 128

Rack1R4#show queueing interface serial 0/1
<snip>
  (depth/weight/total drops/no-buffer drops/interleaves) 1/128/0/0/0
  Conversation 41, linktype: ip, length: 64
  source: 155.1.146.6, destination: 155.1.45.5, id: 0x0046, ttl: 254,
  TOS: 32 prot: 17, source port 58177, destination port 16384

  (depth/weight/total drops/no-buffer drops/interleaves) 19/4048/170/0/0
  Conversation 7, linktype: ip, length: 160
  source: 155.1.146.1, destination: 155.1.108.10, id: 0x4497, ttl: 254, prot: 6
```

The weight value assigned to the VoIP traffic flow is now 128 due to the IP RTP Reserve, which is much better than the weight assigned to data transfer. Also, note the number of the conversation for this reserved flow is 41, which is 32+8+1, meaning that we have 32 dynamic flow queues on the interface.

Now check the IP SLA statistics to confirm the increase in voice quality. Note the improved MOS (Mean Opinion Score) and the decreased RTT value.

```
Rack1R6#show ip sla monitor statistics 1
Round trip time (RTT)      Index 1
       Latest RTT: 18 ms
Latest operation start time: 07:35:33.300 UTC Mon Aug 4 2008
Latest operation return code: OK
RTT Values
       Number Of RTT: 1000
       RTT Min/Avg/Max: 12/18/34 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: 999
       Source to Destination Jitter Min/Avg/Max: 1/4/9 ms
       Destination to Source Jitter Min/Avg/Max: 1/1/12 ms
Packet Loss Values
       Loss Source to Destination: 0           Loss Destination to Source: 0
       Out Of Sequence: 0  Tail Drop: 0 Packet Late Arrival: 0
Voice Score Values
       Calculated Planning Impairment Factor (ICPIF): 11
MOS score: 4.06
Number of successes: 85
Number of failures: 0
Operation time to live: Forever
```


10.4 Legacy RTP Prioritization

- Reset R4's Serial link to R5 to the default fair-queue values, and remove the RTP reservation.
- Configure legacy RTP prioritization for UDP ports in the range 16383 – 32767 up to 128Kbps.

Configuration

```
R4:
interface Serial 0/1
 bandwidth 128
 no hold-queue out
 fair-queue
 max-reserved-bandwidth 100
 no ip rtp reserve
 ip rtp priority 16384 16383 128
```

Verification

Note

The **ip rtp priority** feature was the next natural progression beyond the **ip rtp reserve** feature, as a strict priority queue replaced the RSVP conversation used to schedule the VoIP packets. The IP RTP Priority feature differs from the IP RTP Reserve in that the priority queue has a WFQ weight of zero, meaning that the WFQ always services it first. Packets leaving the priority queue are rate-limited using the configurable bandwidth setting, and use a burst size of 1 second (e.g. for the bandwidth 128Kbps the burst is 128Kbps*1 second = 16 Kbyte). Furthermore, the IP RTP Priority feature performs an additional check to ensure that only *even* UDP port numbers are matched, which increases the VoIP traffic classification quality.

The biggest advantage of the feature is that the voice conversation no longer has to compete with *any* flow, because of its prioritized nature. This is the reason that rate limiting is necessary, to prevent the other queues from being starved.

The command syntax is the same as with IP RTP Reserve, and reads **ip rtp priority *<Starting UDP Port> <Port Range> <Bandwidth>***. Here <*Bandwidth*> *means* the policer rate. As with RTP reserve, the <*Bandwidth*> cannot exceed <*max-reserved-bandwidth*>*<*interface bandwidth*>, which is why the **max-reserved-bandwidth** is changed to 100% in this particular task. Note that the IP RTP priority drops exceeding packets, while IP RTP Reserve simply changes their scheduling weight.

The flow queue number assigned to the priority queue is set right after the special “Link Queue” numbers (used to queue routing updates, L2 keepalives, CDP etc). If there are 2^N dynamic flow queues, the priority queue number is 2^{N+8}. For example, if there are 32 dynamic queues, the priority queue number is 40.

```
Rack1R4#show queueing fair
Current fair queue configuration:

  Interface           Discard    Dynamic  Reserved  Link     Priority
                      threshold  queues   queues    queues   queues
  Serial0/0           64         256      0         8        1
  Serial0/1           64         32       1         8        1
```

The verifications of this feature are the same as with RTP Reserve. R6 sources G.729-like packet streams to R5 through the IP SLA feature, and measure jitter and round-trip time (RTT). At the same time R1 will be configured as an HTTP server and allow other routers to download its IOS image, which will cause additional congestion in the output queue of R4. R6 will mark outgoing packets with an IP Precedence value of 1, and R1 will mark outgoing packets with an IP precedence value of 7 (the maximum priority and minimum WFQ weight). Note that the Frame-Relay interface of R5 is shutdown to make sure R1 prefers to reach R5 through R4, and vice versa.

```
R1:
ip http server
ip http path flash:
!
policy-map MARK
 class class-default
  set ip precedence 7
!
interface FastEthernet 0/0
 service-policy output MARK

R5:
interface Serial 0/0
 shutdown
!
rtr responder

R6:
policy-map MARK
 class class-default
  set ip precedence 1
!
interface FastEthernet 0/0.146
 service-policy output MARK
!
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
```


Start transferring the large file (IOS image) from R1 to SW4.

```
Rack1SW4#copy http://admin:cisco@155.1.146.1/c2600-adventerprisek9-
mz.124-10.bin null:
```

Observe the queue of R4's Serial interface.

```
Rack1R4#show queue serial 0/1
<snip>

  (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: 0x01A0, ttl: 254,
  TOS: 32 prot: 17, source port 57143, destination port 16384

  (depth/weight/total drops/no-buffer drops/interleaves) 32/4048/0/0/0
  Conversation 7, linktype: ip, length: 160
  source: 155.1.146.1, destination: 155.1.108.10, id: 0x9280, ttl: 254, prot: 6

Rack1R4#show interfaces serial 0/1
Serial0/1 is up, line protocol is up
  Hardware is PowerQUICC Serial
  Internet address is 155.1.45.4/24
<snip>
  30 second input rate 38000 bits/sec, 84 packets/sec
  30 second output rate 121000 bits/sec, 139 packets/sec
```

Note the special "conversation 40", which has no weight value assigned. This is the IP RTP priority queue. The other flow is the HTTP traffic. Now verify the probe statistics on R6.

```
Rack1R6#show ip sla monitor statistics 1
Round trip time (RTT)      Index 1
       Latest RTT: 18 ms
Latest operation start time: 08:15:49.305 UTC Mon Aug 4 2008
Latest operation return code: OK
RTT Values
       Number Of RTT: 1000
       RTT Min/Avg/Max: 13/18/33 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: 999
       Source to Destination Jitter Min/Avg/Max: 1/5/10 ms
       Destination to Source Jitter Min/Avg/Max: 1/1/15 ms
Packet Loss Values
       Loss Source to Destination: 0           Loss Destination to Source: 0
       Out Of Sequence: 0  Tail Drop: 0 Packet Late Arrival: 0
Voice Score Values
       Calculated Planning Impairment Factor (ICPIF): 11
MOS score: 4.06
Number of successes: 29
Number of failures: 0
```

