Queuing

Overview

First In, First Out (FIFO)

Round Robin & Weighted Round Robin (WRR)

Priority Queuing (PQ)

Weighted Fair Queuing (WFQ)

Class-Based Weighted Fair Queuing (CBWFQ)

Low Latency Queuing (LLQ)

Link Fragmenting & Interleaving (LFI)

"Hot Spots And Gotchas"

At its core, queuing is a congestion management technique that allows us a degree of control over which packets a router will transmit first.

Basically, we're doing three things - creating the queues, deciding which traffic goes into each queue by classifying the traffic, and then deciding the order in which queues will be allowed to send their traffic ("scheduling").

Of course, you and I both know it gets just a *bit* more complex. We've got quite a few queueing options, and the first one is the most basic - FIFO.

First-In, First-Out

If ever there's been a "the name is the recipe" networking term, this is it. Traffic isn't ranked, marked, classified, or anything else - it's just queued up and sent, and traffic is queued in the order in which it arrived.

The default queuing scheme for interfaces running at greater than E1 speed, FIFO is just fine in some instances, but the amount of time-

sensitive data on today's networks is increasing every day. Voice and video are particularly subject to jitter, and if you've ever tried to watch streaming video on a slow connection, you know how frustrating that can be.

Also, every organization has applications that are more critical to operations than others. Yes, I know that every end user thinks their apps are the most important, but some apps truly are more important than others! You don't want traffic created by your company's critical apps to sit in a FIFO queue with other, less-important traffic.

I was once the network admin for a hospital, and there were applications that helped save lives - and if those apps were down, patients' lives were at risk. The data produced by those apps could not be left to FIFO.

Luckily for us, Cisco routers give us quite a few options when it comes to queuing, and there's a pretty good chance we'll see a lot of these options on the ONT exam.

Round-Robin And Weighted Round-Robin (WRR)

With *round-robin* (RR) queuing, no one queue is given priority over another. For example, if you have five queues in an RR scheme, one packet will be sent from Queue 1, then one from Queue 2, and so forth until each queue has had an opportunity to send a packet. The process then repeats itself. If you've worked with *Custom Queuing* (CQ) before, that's an excellent example of RR queuing. If you haven't worked with CQ before, stick around!

Weighted Round-Robin (WRR) allows you to assign *weights* to queues; queues that have higher weights will transmit more packets than others, but it's still a round-robin format.

Let's say we have five queues and the following weights:

Q1: 5 Q2: 4 Q3: 3 Q4: 2 Q5: 1

Theoretically, Q1 will send five packets, then Q2 will send four, Q3 will send three, Q4 will send two, and Q5 will send one. Every queue gets a chance to transmit, but again the queues with higher weights get to send more packets.

That's the theory of how WRR works. You probably won't run into a scenario like the following until you're going after your CCIE number, but let's take a look at how Custom Queuing (CQ) is configured, and work in a situation where WRR isn't quite so cut-and-dried.

Using Weights With Custom Queueing

In this example, we'll use source IP addresses to decide which queue traffic should be placed into.

- Q1: Source 172.12.123.0 /24 Weight: 3
- Q2: Source 210.1.1.0 /24 Weight: 2
- Q3: Source 215.1.1.0 /24 Weight: 1
- Q4: All traffic that does not match any of the above. Weight: 1

As you've probably guessed, we first have to write ACLs matching those three definitions.

```
R1(config)#access-list 10 permit 172.12.123.0 0.0.0.255
R1(config)#access-list 20 permit 210.1.1.0 0.0.0.255
R1(config)#access-list 30 permit 215.1.1.0 0.0.0.255
```

We'll use the *queue-list* command to match those ACLs to the appropriate queues. I'll use IOS Help to show the options for the first queue-list. The syntax is a little tricky when you first work with CQ, but you'll quickly get used to it.

```
R1(config)#queue-list 1 ?
  default        Set custom queue for unspecified datagrams
  interface      Establish priorities for packets from a named interfac
  lowest-custom  Set lowest number of queue to be treated as custom
  protocol       priority queueing by protocol
  queue          Configure parameters for a particular queue
  stun           Establish priorities for stun packets

R1(config)#queue-list 1 protocol ?
  arp            IP ARP
  bridge         Bridging
  cdp            Cisco Discovery Protocol
  compressedtcp  Compressed TCP
  ip             IP
  llc2           llc2
  pad            PAD links
  snapshot       Snapshot routing support

R1(config)#queue-list 1 protocol ip ?
  <0-16>  queue number

R1(config)#queue-list 1 protocol ip 1 ?
  fragments  Prioritize fragmented IP packets
  gt         Classify packets greater than a specified size
  list       To specify an access list
  lt         Classify packets less than a specified size
  tcp        Prioritize TCP packets 'to' or 'from' the specified port
  udp        Prioritize UDP packets 'to' or 'from' the specified port
  <cr>

R1(config)#queue-list 1 protocol ip 1 list ?
  <1-199>      IP access list
  <1300-2699>  IP expanded access list

R1(config)#queue-list 1 protocol ip 1 list 10
```

```
R1(config)#queue-list 1 protocol ip 2 list 20
R1(config)#queue-list 1 protocol ip 3 list 30
```

Like ACLs, queue-lists are read from top to bottom. Once a match is made, the traffic is queued and no other lines of the queue-list are considered.

Now we'll add a default queue that will match all traffic that does not match the first three lines of the queue-list.

```
R1(config)#queue-list 1 default 4
```

Now you're thinking, "That's all great, but what about the weights?" We can assign a type of weight to each queue by assigning a *byte-count* to each queue. Let's review the desired weights:

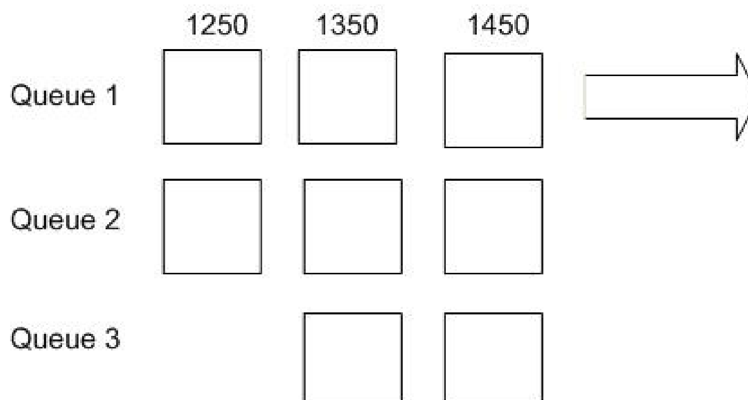
- Q1: Source 172.12.123.0 /24 Weight: 3
- Q2: Source 210.1.1.0 /24 Weight: 2
- Q3: Source 215.1.1.0 /24 Weight: 1

Q4: All traffic that does not match any of the above. Weight: 1

That means Q1's byte-count should be three times that of Q3 and Q4, and Q2's byte-count should be twice that of Q3 and Q4. One configuration that would give us the desired result is:

```
R1(config)#queue-list 1 queue 1 byte-count 3000
R1(config)#queue-list 1 queue 2 byte-count 2000
R1(config)#queue-list 1 queue 3 byte-count 1000
R1(config)#queue-list 1 queue 4 byte-count 1000
```

Pretty easy, eh? :) It really is simple. The only issue arises from the fact that not all packets are the same size, and *CQ will continue to transmit packets as long as that byte-count hasn't been reached yet*. Consider this example:



Q1 has been configured with a byte-count of 3000 bytes. The first two packets are 1450 and 1350 bytes, for a total of 2800 bytes. The question is - will the 1250-byte packet in Q1 be transmitted, or will Q2 now begin to transmit?

Q1 will indeed transmit the 1250-byte packet. CQ considers the overall byte-count after each packet is sent - and if the queue is still under its byte-count limit, the queue will send the next packet in full, *even if that packet puts the queue over its byte-count limit.* The 1250-byte packet in this example will not be fragmented - it will be transmitted in its entirety.

This is hardly the end of the world, but it's a good detail to keep in mind. CQ weights aren't perfect, but they are effective.

If you want to bypass the byte-count calculations and give each queue a strict packet limit instead, you can do so with the queue-list command's *limit* option. Here, we'll set the default queue's limit to 100 packets after removing the byte-count command.

```
R1(config)#no queue-list 1 queue 4 byte-count 1000
R1(config)#queue-list 1 queue 4 ?
  byte-count  Specify size in bytes of a particular queue
 limit      Set queue entry limit of a particular queue

R1(config)#queue-list 1 queue 4 limit ?
  <0-32767>  number of queue entries

R1(config)#queue-list 1 queue 4 limit 100
R1(config)#
```

Hey, after all that, we should apply the list to the interface, right? We'll do so with the *custom-queue-list* command. There are no options with this command.

We'll verify all of this with *show queueing custom.* This command displays the queue-list numbers, the queue numbers, and everything else you need to know about your CQ configuration, including any ACLs in use.

```
R1#show queueing custom
Current custom queue configuration:

List   Queue  Args
1      4      default
1      1      protocol ip          list 10
1      2      protocol ip          list 20
1      3      protocol ip          list 30
1      1      byte-count 3000
1      2      byte-count 2000
```

```
1       3       byte-count 1000
1       4       limit 100
```

Priority Queuing

PQ is simple enough to configure, but there's one big trap you have to watch out for when configuring it. Before we discuss that, let's go over the basics of PQ.

PQ has four and only four queues. All four are predefined as to priority and capacity. Note that the Normal queue is the default queue; packets that have not had a priority explicitly assigned to them are placed into that queue.

High-Priority Queue: Capacity of 20 packets

Medium-Priority Queue: Capacity of 40 packets

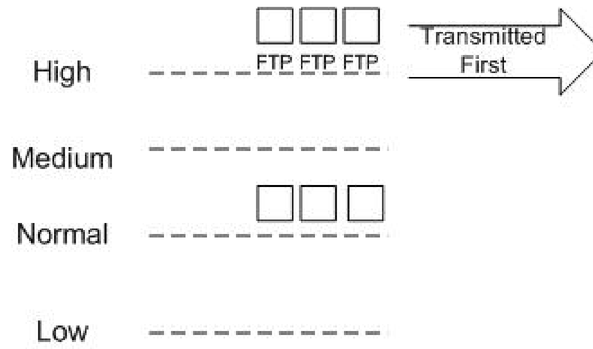
Normal-Priority (Default) Queue: Capacity of 60 packets

Low-Priority Queue: Capacity of 80 packets

As you'll see in just a moment, changing the capacity of any of these queues is easy. The difficult part of working with PQ is resisting the temptation to configure a lot of traffic as high priority (yeah, I know, *everybody's* traffic is high priority - just like their email, right?).

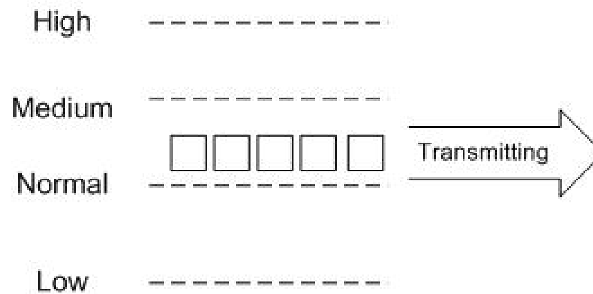
Why is this a problem? PQ does *not* work in a round-robin format, as some other queuing strategies do. Regardless of how much traffic is waiting in the lower queues, the High-priority queue is always going to be given first priority, and that means traffic in the lower-priority queues can sit there for a *long* time. Let's take a look at one basic scenario that illustrates this.

As the network admin, you decide that FTP packets should be given the highest priority possible. You configure FTP packets to be placed in the High-priority queue, and they're transmitted before any other traffic. Since you've defined no other priorities, all other traffic is placed into the Normal queue.

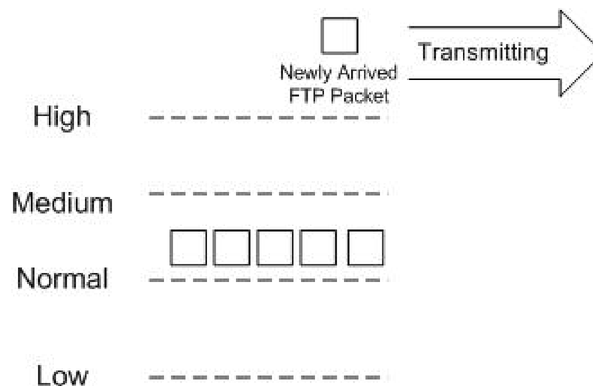


This in itself would probably not cause a problem, but the temptation is to give too many other traffic types a higher priority as well. If too many traffic types are placed into the higher queues, packets can end up sitting in the lower queues for too long.

With PQ, anytime a packet enters the High queue, the router will stop transmitting any other queue's traffic and transmit the high-priority traffic. The router can be transmitting traffic normally, in this case from the Normal queue...



... but it only takes a single packet to arrive on a higher-priority queue for the router to stop doing so and transmit the newly-arrived packet, while the packets in the other queues that actually arrived earlier sit and wait their turn. And if too many packets enter those higher-priority queues, that turn may not come for quite a while!



This is referred to as *packet starvation* and *queue starvation*, but no

matter what you call it, it's bad! The first step toward preventing this scenario is resisting the temptation to define too many different traffic types as high or medium priority.

And how do you define traffic with PQ? Glad you asked!

Step One: Create The Priority List

Remember how you wrote dialer lists in your CCNA studies to define interesting traffic? Creating priority lists in PQ is a similar operation, but the priority list will actually do two things - define traffic, and define which queue that traffic named by the list should be placed into. If traffic is not explicitly placed into a queue, it will be placed into the Normal queue.

Since we're studying Cisco, you just know we're going to have quite a few options! Let's use IOS Help to view the options for priority lists:

```
R1(config)#priority-list ?
  <1-16>  Priority list number
```

This list number has the same purpose as the dialer list number in your CCNA studies - this is the number that must be used in the interface-level command in order to put this list into effect.

```
R1(config)#priority-list 1 ?
  default      Set priority queue for unspecified datagrams
  interface    Establish priorities for packets from a named interface
  protocol     priority queueing by protocol
  queue-limit  Set queue limits for priority queues
```

The first option, *default,* allows us to change the default queue priority from Normal. We'll use this command to make the Low queue the default queue.

```
R1(config)#priority-list 1 default ?
  high
  medium
  normal
  low

R1(config)#priority-list 1 default low
```

Always trust your configs, but verify them as well. We'll use *show queueing priority* throughout this section to perform that verification. Here, we see that the Low queue is now the default queue for priority list 1.

```
R1#show queueing priority
Current DLCI priority queue configuration:
Current priority queue configuration:

List   Queue  Args
1      low    default
```

Let's take another look at the priority-list command.

```
R1(config)#priority-list 1 ?
  default      Set priority queue for unspecified datagrams
  interface    Establish priorities for packets from a named interface
  protocol     priority queueing by protocol
  queue-limit  Set queue limits for priority queues
```

The middle two options are key. We can apply this queuing scheme on a per-interface level or a per-protocol level. If we choose the per-interface approach, note that we can use physical or logical interfaces to do so.

```
R1(config)#priority-list 1 interface ?
  Async              Async interface
  BRI                ISDN Basic Rate Interface
  BVI                Bridge-Group Virtual Interface
  CTunnel            CTunnel interface
  Dialer             Dialer interface
  Ethernet           IEEE 802.3
  Group-Async        Async Group interface
  Lex                Lex interface
  Loopback           Loopback interface
  Multilink          Multilink-group interface
  Null               Null interface
  Serial             Serial
  Tunnel             Tunnel interface
  Vif                PGM Multicast Host interface
  Virtual-Template   Virtual Template interface
  Virtual-TokenRing  Virtual TokenRing
```

We'll configure all traffic *coming in* on this router's Serial0 interface to go into the Normal queue. This does not affect traffic exiting the router via Serial0.

```
R1(config)#priority-list 1 interface serial 0 ?
  high
  medium
  normal
  low

R1(config)#priority-list 1 interface serial 0 normal ?
  <cr>
```

When IOS Help shows the only option is <CR>, that means we're out of options! After entering this command, we'll verify with show queueing priority.

```
R1(config)#priority-list 1 interface serial 0 normal

R1#show queueing priority
Current DLCI priority queue configuration:
Current priority queue configuration:

List   Queue  Args
1      low    default
1      normal interface Serial0
```

When using the protocol option, the syntax of the command can be a little

awkward at first. To illustrate, we'll write a line placing all TCP traffic into the Medium queue. Let's use IOS Help to figure out how to do so.

```
R1(config)#priority-list 1 protocol ?
  aarp               AppleTalk ARP
  appletalk          AppleTalk
  arp                IP ARP
  bridge             Bridging
  cdp                Cisco Discovery Protocol
  compressedtcp      Compressed TCP
  decnet             DECnet
  decnet_node        DECnet Node
  decnet_router-l1   DECnet Router L1
  decnet_router-l2   DECnet Router L2
  ip                 IP
  ipx                Novell IPX
  llc2               llc2
  pad                PAD links
  snapshot           Snapshot routing support
```

Hmmm. I've highlighted IP in this list, but that covers a lot of territory! How can we define TCP traffic in this list, since it's not listed here? Actually, TCP traffic is an option -- at the *end* of the command.

```
R1(config)#priority-list 1 protocol ip medium ?
  fragments  Prioritize fragmented IP packets
  gt         Prioritize packets greater than a specified size
  list       To specify an access list
  lt         Prioritize packets less than a specified size
  tcp        Prioritize TCP packets 'to' or 'from' the specified port
  udp        Prioritize UDP packets 'to' or 'from' the specified port
  <cr>
```

Like I always say, IOS Help is a lifesaver! If we wanted to specify a certain TCP port for PQ, the port number can be named as well.

```
R1(config)#priority-list 1 protocol ip medium tcp ?
  <0-65535>    Port number
  bgp          Border Gateway Protocol (179)
  chargen      Character generator (19)
  cmd          Remote commands (rcmd, 514)
  daytime      Daytime (13)
```

The router lists about 50 different port numbers at that point, so I'm only showing you a few. We'll choose port 13 to place Daytime packets into the Medium queue - unless we have even more options.

```
R1(config)#priority-list 1 protocol ip medium tcp 13 ?
  <cr>
```

Nope! Just hit <enter> and Daytime packets will then be placed into the Medium queue. Verify with show queueing priority.

```
R1#show queueing priority
Current DLCI priority queue configuration:
Current priority queue configuration:

List   Queue  Args
```

```
1      low    default
1      normal interface Serial0
1      medium protocol ip          tcp port daytime
```

If you really want to be specific, you can use the gt and lt options to specify queuing for packets of certain size. You can also define the queue that fragmented packets should be placed into.

```
R1(config)#priority-list 1 protocol ip medium ?
  fragments  Prioritize fragmented IP packets
  gt         Prioritize packets greater than a specified size
  list       To specify an access list
  lt         Prioritize packets less than a specified size
  tcp        Prioritize TCP packets 'to' or 'from' the specified port
  udp        Prioritize UDP packets 'to' or 'from' the specified port
  <cr>
```

That's probably going to be too specific for most queuing strategies, but you may well want to incorporate access lists when using PQ. Say you wanted to place all traffic from the private network 10.0.0.0 /8 into the Normal queue. An ACL is used to name that network - you could also specify the destination if you use an extended ACL - and the list is then named by the priority list.

```
R1(config)#access-list 10 permit 10.0.0.0 0.0.0.255

R1(config)#priority-list 1 protocol ip normal ?
  fragments  Prioritize fragmented IP packets
  gt         Prioritize packets greater than a specified size
  list       To specify an access list
  lt         Prioritize packets less than a specified size
  tcp        Prioritize TCP packets 'to' or 'from' the specified port
  udp        Prioritize UDP packets 'to' or 'from' the specified port
  <cr>

R1(config)#priority-list 1 protocol ip normal list ?
  <1-199>      IP access list
  <1300-2699>  IP expanded access list

R1(config)#priority-list 1 protocol ip normal list 10
```

You know the drill from here on out.... verify, verify, verify!

```
R1#show queueing priority
Current DLCI priority queue configuration:
Current priority queue configuration:

List   Queue  Args
1      low    default
1      normal interface Serial0
1      medium protocol ip          tcp port daytime
1      normal protocol ip          list 10
```

After all of this work, we should apply the list to an interface! We'll use the *priority-group* command to apply this list to interface serial0. It may surprise you to find that there are no options with this command!

```
R1(config)#interface serial0
R1(config-if)#priority-group 1 ?
  <cr>

R1(config-if)#priority-group 1
```

Weighted Fair Queuing

The default queuing scheme for *Serial* interfaces running at E1 speed or below, WFQ handles packets according to their *flow,* with a "flow" defined as packets that have one of the following in common:

- Source or destination IP address
- Source or destination port number (TCP or UDP)
- Protocol Number (that's why I mention them occasionally)
- ToS (Type of Service)

Therefore, a group of packets destined for the IP address 213.1.1.1 would be considered to be part of the same flow.

The key word in WFQ is "fair", since "fair" isn't a term we see in networking very often! WFQ is considered "fair" because it assigns the same weight to all traffic flows, while at the same time giving priority to low-volume, interactive flows over high-volume flows ("aggressive flows", in Cisco website documentation terms). This priority results in packets being dropped from aggressive flows before they're dropped from low-volume flows.

A major difference between WFQ and other queuing strategies is that WFQ will dynamically build and tear down queues as they are needed. WFQ cannot build an infinite number of queues, however; the default maximum number of dynamic WFQ queues is 256. This can be changed, but you can't just change it to any old number - that would be too easy!

Before we set the number of reservable queues, let's assume this serial interface is running at over E1 speed - which means it would not be running WFQ by default. Here's how you enable WFQ:

```
R1(config)#int serial 0
R1(config-if)#fair-queue
```

Now let's set the number of dynamic queues.

```
R1(config)#int serial 0
R1(config-if)#fair-queue ?
  <1-4096>  Congestive Discard Threshold
  <cr>

R1(config-if)#fair-queue 100 ?
  <16-4096>  Number Dynamic Conversation Queues
```

```
  <cr>

R1(config-if)#fair-queue 100 200
Number of dynamic queues must be a power of 2 (16, 32, 64, 128, 256, 512,
1024)
```

I tried to set the number of Dynamic Conversation Queues to 200, but Cisco routers will not allow that. The router is kind enough to tell us that the number must be a power of 2!

Note in the above config that I had to set the Congestive Discard Threshold (CDT) before I could set the number of Dynamic Conversation Queues. The CDT is the number of packets a queue can hold before WFQ will start to drop packets from high-volume conversations.

A third value that can be set with the *fair-queue* command is the number of Reservable Queues (RQ). The RQ value is, well, the number of queues reserved for reserved conversations! If that sounds like a feature that our old friend Resource Reservation Protocol (RSVP) would use, you're right. RSVP reserves bandwidth for a transmission from source to destination before sending the transmission, and RSVP uses these queues when WFQ is in effect.

By default, there are no RQs. To create them, use the *fair-queue* command as shown:

```
R1(config-if)#fair-queue 100 64 ?
  <0-1000>  Number Reservable Conversation Queues
  <cr>

R1(config-if)#fair-queue 100 64 100
```

We skipped around a bit on that command, so let's review from left to right:

The first value is the Congestive Discard Threshold. Remember that WFQ will drop packets from high-volume conversations ("aggressive flows") first; the CDT is the number of packets that will be held in a queue before packets are dropped from those conversations. Default is 64.

The second value is the number of Dynamic Conversation Queues. The default is 256, and when changing this value, you must set it to a power of 2.

The third and final value is the number of Reservable Queues. The default is zero, and the range is 0 - 1000.

Real-World vs. Theory In WFQ

I want to bring something to your attention regarding WFQ theory and its actual operation. In the previous config, we set the Congestive Discard

Threshold to 100 with no problem. According to Cisco and non-Cisco documentation, that value is required to be set to a power of 2, just as the number of Dynamic Queues was. That wasn't the case on this router, though.

Let's set it to 17 just to make sure.

```
R1(config-if)#fair-queue ?
  <1-4096>  Congestive Discard Threshold
  <cr>

R1(config-if)#fair-queue 17
```

On rare occasions, a Cisco router will accept an illegal command without prompting the admin, but the command doesn't actually take effect. Let's verify that command's results with *show queueing interface.*

```
R1#show queueing interfac serial 0/0/0
Interface Serial0/0/0 queueing strategy: fair
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: weighted fair
  Output queue: 0/1000/17/0 (size/max total/threshold/drops)
     Conversations  0/0/256 (active/max active/max total)
     Reserved Conversations 0/0 (allocated/max allocated)
     Available Bandwidth 1158 kilobits/sec
```

Hmm. I see "17" there. Let's double-verify with *show queueing.*

```
R1#show queueing
Current fair queue configuration:

  Interface     Discard    Dynamic  Reserved  Link    Priority
                threshold  queues   queues    queues  queues
  Serial0         17        256       0        8         1
```

Sure looks like the non-Base2 value was accepted. :) I doubt this comes up on your ONT exam, but just in case it does, I'd stick with the official Cisco theory that both the Congestive Discard Threshold and number of Dynamic Queues must both be set to a power of 2. As we well know, theory doesn't always match up with the real world!

The Exception To The WFQ Rule

I mentioned several times that WFQ is the default queueing scheme for Serial interfaces running at or below E1 speed, but as we all know, there are always exceptions. WFQ cannot serve as the default for serial interfaces using any of the following encap methods or interface types, regardless of the interface's speed:

- Virtual interfaces, including loopbacks and dialer interfaces
- Bridging or tunneling

- LAPB, X.25, SDLC

Disabling And Enabling WFQ

WFQ's the default queuing scheme for Serial interfaces running at or less than E1 speed, but what if you want to disable it? Or reenable it? Just use the *no fair-queue* and *fair-queue* commands, respectively.

```
R1(config)#int s0
R1(config-if)#no fair-queue

R1(config-if)#fair-queue
Must remove custom-queue configuration first.
```

Don't forget - you can have only one queueing scheme running on a single interface at one time! The router was kind enough to remind us of that, but the ONT exam will likely not be as kind.

```
R1(config-if)#no custom-queue-list 1
R1(config-if)#fair-queue
```

Class-Based Weighted Fair Queuing

With CBWFQ, we're going to create classes of traffic, and these classes are each assigned their own queue. The queues can then be assigned a guaranteed amount of bandwidth.

Doesn't exactly sound "fair", does it? The key is that each queue is going to be guaranteed a certain amount of available bandwidth. In our PQ discussion, I mentioned several times that you have to watch for the possibility of queue starvation, but that danger doesn't exist with CBWFQ, since every queue is guaranteed some bandwidth.

Some important details regarding CBWFQ:

- You can create up to 64 queues
- The queues themselves are FIFO queues, but you can configure the queues with WRED, as we'll soon see
- Traffic that is not explicitly placed in a queue is placed into the default queue, appropriately named *class-default.*

Configuring CBWFQ

The first step in configuring CBWFQ is to identify the traffic that should be

placed in a given queue. We'll take all UDP Voice ports and put them into one queue, while all HTTP traffic will be placed into another queue. Any remaining traffic will be placed into the default queue. We'll identify those two groups of traffic with access lists. (WFQ doesn't use ACLs, but CBWFQ does.)

```
R1(config)#access-list 110 permit udp any any range 16384 32767
R1(config)#
R1(config)#
R1(config)#access-list 120 permit tcp any any eq 80
```

That's not a bad range of UDP ports to know, by the way. :)

Now that we've used ACLs to identify the traffic, we'll use class maps that will be used to match those two ACLs. We'll use the Modular Command-line Interface (MQC) to create class maps.

After creating the class map UDPVOICE and configuring it to match traffic defined by ACL 110, I'll run IOS Help so you can see the other options. That will be followed by the creation of the class map HTTP, which will match traffic defined by ACL 120.

```
R1(config)#class-map UDPVOICE
R1(config-cmap)#match access-group 110
R1(config-cmap)#?
QoS class-map configuration commands:
  description  Class-Map description
  exit         Exit from QoS class-map configuration mode
  match        classification criteria
  no           Negate or set default values of a command
  rename       Rename this class-map

R1(config)#class-map HTTP
R1(config-cmap)#match access-group 120
```

You're not limited to ACLs for matching. IOS Help shows that we have *plenty* of other options, including *input-interface* and *protocol.*

```
R1(config-cmap)#match ?
  access-group          Access group
  any                   Any packets
  class-map             Class map
   cos                    IEEE  802.1Q/ISL  class  of  service/user  priority
values
  destination-address  Destination address
  discard-class         Discard behavior identifier
  dscp                  Match DSCP in IP(v4) and IPv6 packets
  fr-de                 Match on Frame-relay DE bit
  fr-dlci               Match on fr-dlci
  input-interface       Select an input interface to match
  ip                    IP specific values
  mpls                  Multi Protocol Label Switching specific values
  not                   Negate this match result
  packet                Layer 3 Packet length
  precedence            Match Precedence in IP(v4) and IPv6 packets
  protocol              Protocol
  qos-group             Qos-group
```

```
  source-address       Source address
```

We're now going to define the parameters of the queues with a policy map. Before we configure the values for the above traffic classes, let's use IOS Help to see our options.

```
R1(config)#policy-map BRYANTPOLICY
R1(config-pmap)#?
QoS policy-map configuration commands:
  class         policy criteria
  description   Policy-Map description
  exit          Exit from QoS policy-map configuration mode
  no            Negate or set default values of a command
  rename        Rename this policy-map
```

Not much to choose from! We'll use the *class* command here and name the class we want to define values for.

```
R1(config-pmap)#class UDPVOICE
R1(config-pmap-c)#?
QoS policy-map class configuration commands:
  bandwidth         Bandwidth
  compression       Activate Compression
  drop              Drop all packets
  estimate          estimate resources required for this class
  exit              Exit from QoS class action configuration mode
  netflow-sampler   NetFlow action
  no                Negate or set default values of a command
  police            Police
  priority          Strict Scheduling Priority for this Class
  queue-limit       Queue Max Threshold for Tail Drop
  random-detect     Enable Random Early Detection as drop policy
  service-policy    Configure Flow Next
  set               Set QoS values
  shape             Traffic Shaping
```

Now that's a little more like it, right? Let's take a look at the most commonly-used values with policy maps, along with a few little "gotchas".

```
R1(config-pmap-c)#bandwidth ?
  <8-2000000>  Kilo Bits per second
  percent      % of total Bandwidth
  remaining    % of the remaining bandwidth
```

We've actually got three options with the *bandwidth* command. The first option calls for a numeric value - note that this value is in kbps!

The second option, *percent*, defines the percentage of available overall bandwidth that will be assigned to that particular class. Both WFQ and CBWFQ have the bandwidth percentage options.

The third option, *remaining,* also defines the percentage of bandwidth that should be assigned to this class, but *it's the percentage of bandwidth that has not yet been assigned to other classes.* This is a relatively new command, so even if you've worked with CBWFQ before, you may not have seen that one.

There's one more detail with bandwidth that we have to watch out for - we can't assign more than 75% of available bandwidth. The router will reserve 25% of available bandwidth for network control traffic and routing overhead. This is a good thing - after all, we don't want to cut off our routing control traffic, or we're not going to have routing!

If you really feel the need to override this percentage, use the *max-reserved-bandwidth* command. Don't blame me for all the hyphens!

```
R1(config-if)#max-reserved-bandwidth ?
  <1-100>  Max. reservable bandwidth as % of interface bandwidth
```

Okay, I lied. There is another detail involving the bandwidth command! There is no cut-and-dried situation where you should definitely define bandwidth in kbps or with the *bandwidth percent* command, but Cisco routers don't like it when you try to use both in the same policy map. Here, we'll set the bandwidth to 25 kbps for the UDPVOICE class. Remember that when we get to the HTTP class!

```
R1(config)#policy-map BRYANTPOLICY
R1(config-pmap)#class UDPVOICE
R1(config-pmap-c)#bandwidth 25
All classes with bandwidth should have consistent units
```

There's a friendly little reminder from the router as well!

You'll be happy to know that other important *policy-map* commands don't have as many options. The *queue-limit* command does exactly what it sounds like it does - it sets the packet limit for the queue. We'll set the limit here to 512 packets.

```
R1(config-pmap-c)#queue-limit ?
  <1-4096>  Packets

R1(config-pmap-c)#queue-limit 512
```

Before we configure the HTTP class, let's verify what we've done so far with *show policy-map*.

```
R1#show policy-map
  Policy Map BRYANTPOLICY
    Class UDPVOICE
      Bandwidth 25 (kbps) Max Threshold 64 (packets)
```

So far, so good. Let's configure the HTTP class with a bandwidth percentage of 10 and a queue-limit of 100 packets.

```
R1(config)#policy-map BRYANTPOLICY
R1(config-pmap)#class HTTP
R1(config-pmap-c)#bandwidth percent 10
All classes with bandwidth should have consistent units
```

```
R1(config-pmap-c)#queue-limit 100
bandwidth on the class is required to issue this command
```

We ran into all kinds of trouble with that one! After that last message, I did a ctrl-Z and a save, so let's see what saved:

```
R1#show policy-map
  Policy Map BRYANTPOLICY
    Class UDPVOICE
      Bandwidth 25 (kbps) Max Threshold 64 (packets)
    Class HTTP
```

Not much! Before you configure a policy map, you've got to decide whether you're going to define bandwidth explicitly or as a percentage, because you can't do both!

```
R1(config)#policy-map BRYANTPOLICY
R1(config-pmap)#class HTTP
R1(config-pmap-c)#bandwidth 10
R1(config-pmap-c)#queue-limit 100
```

Always verify your config with the appropriate *show* command.

```
R1#show policy-map
  Policy Map BRYANTPOLICY
    Class UDPVOICE
      Bandwidth 25 (kbps) Max Threshold 64 (packets)
    Class HTTP
      Bandwidth 10 (kbps) Max Threshold 100 (packets)
```

Finally, we'll define some values for the default queue. The options are the same:

```
R1(config-pmap)#class class-default
R1(config-pmap-c)#?
QoS policy-map class configuration commands:
  bandwidth         Bandwidth
  compression       Activate Compression
  drop              Drop all packets
  estimate          estimate resources required for this class
  exit              Exit from QoS class action configuration mode
  fair-queue        Enable Flow-based Fair Queuing in this Class
  netflow-sampler   NetFlow action
  no                Negate or set default values of a command
  police            Police
  priority          Strict Scheduling Priority for this Class
  queue-limit       Queue Max Threshold for Tail Drop
  random-detect     Enable Random Early Detection as drop policy
  service-policy    Configure Flow Next
  set               Set QoS values
  shape             Traffic Shaping
```

The default packet drop method for the default class is tail drop, but we can easily change that. To enable RED for the default class, just use the *random-detect* command...

```
R1(config-pmap-c)#random-detect
fair-queue or bandwidth on the class is required to issue this command
```

... right after you enable WFQ or set a bandwith value! Here, we'll enable WFQ and set the number of Dynamic Queues to 32.

```
R1(config-pmap-c)#fair-queue 32
R1(config-pmap-c)#random-detect
```

As you can see, there are a lot of options with CBWFQ! The key for real-world success with CBWFQ is to plan your queuing strategy in advance - don't just sit down at the router and wing it.

Hey, we ought to apply this policy map! We do so with the *service-policy* command. IOS Help will show options for input and output policies, but let's think about this - can we really have a queuing policy for incoming packets?

```
R1(config)#int ser 0
R1(config-if)#service-policy ?
  history  Keep history of QoS metrics
  input    Assign policy-map to the input of an interface
  output   Assign policy-map to the output of an interface

R1(config-if)#service-policy input ?
  WORD  policy-map name

R1(config-if)#service-policy input BRYANTPOLICY
CBWFQ : Can be enabled as an output feature only
R1(config-if)#service-policy output BRYANTPOLICY
```

No. :) CBWFQ policies can only be put into effect for outbound packets.

Verify the overall config with *show policy-map interface.*

```
R1#show policy-map interface s0/0/0

 Serial0/0/0

  Service-policy output: BRYANTPOLICY

    Class-map: UDPVOICE (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: access-group 110
      Queueing
        Output Queue: Conversation 41
        Bandwidth 25 (kbps)Max Threshold 64 (packets)
        (pkts matched/bytes matched) 0/0
        (depth/total drops/no-buffer drops) 0/0/0

    Class-map: HTTP (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: access-group 120
      Queueing
        Output Queue: Conversation 42
        Bandwidth 10 (kbps)Max Threshold 100 (packets)
        (pkts matched/bytes matched) 0/0
        (depth/total drops/no-buffer drops) 0/0/0
```

```
  Class-map: class-default (match-any)
    0 packets, 0 bytes
    5 minute 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
       exponential weight: 9
```

Low Latency Queuing

Waaaay back at the beginning of this chapter, I mentioned that a major reason we don't stick with FIFO is that today's networks handle more Voice traffic than ever before. Voice traffic needs the highest priority we can give it, and one drawback of CBWFQ is that while we do have a default queue, that default queue isn't given higher priority than the others.

Configuring LLQ creates such a queue. This *strict priority queue* is created primarily for Voice traffic, which is much more sensitive to jitter and delay than "regular" data traffic.

When we configure LLQ in just a moment, it's going to look a great deal like CBWFQ. Basically, LLQ is an add-on or extension to CBWFQ. The commands are almost all the same, and LLQ uses the *class-default* class just as CBWFQ does. The major difference between LLQ and CBWFQ is the creation of that strict priority queue, and there's one simple word that creates that queue - *priority*.

We'll start this example with an ACL defining the UDP voice ports. I told you this was a good group of ports to know!

```
R1(config)#access-list 110 permit udp any any range 16384 32767
```

Just as before, we'll use the class-map command to match ACL 110.

```
R1(config)#class-map VOICE
R1(config-cmap)#match access-group 110
```

Here's where the strict priority queue comes in. In the policy map configuration, we'll use the *priority* command to assign bandwidth rather than the *bandwidth* command.

```
R1(config)#policy-map EXAMPLE
R1(config-pmap)#class VOICE
R1(config-pmap-c)#?
QoS policy-map class configuration commands:
  bandwidth         Bandwidth
  compression       Activate Compression
  drop              Drop all packets
  estimate          estimate resources required for this class
  exit              Exit from QoS class action configuration mode
```

```
  netflow-sampler  NetFlow action
  no               Negate or set default values of a command
  police           Police
  priority       Strict Scheduling Priority for this Class
  queue-limit      Queue Max Threshold for Tail Drop
  random-detect    Enable Random Early Detection as drop policy
  service-policy   Configure Flow Next
  set              Set QoS values
  shape            Traffic Shaping

R1(config-pmap-c)#priority ?
  <8-2000000>  Kilo Bits per second
  percent      % of total bandwidth

R1(config-pmap-c)#priority percent 25
```

The rules for the *priority* command in LLQ are the same as they are for the bandwidth command in CBWFQ - you can't use the *priority* and *priority percent* commands in the same policy map. Here, we assigned 25% of the overall available bandwidth to the strict priority queue. After defining the priority queue's parameters, you can then create other classes just as we did with CBWFQ.

The policy map is applied in the same fashion as CBWFQ, and the policy map can only be applied to outgoing traffic.

```
R1(config-if)#service-policy output EXAMPLE
```

I won't put the entire output of *show policy-map* here, but note that the strict priority queue creation is verified with this command.

```
Service-policy output: EXAMPLE

  Class-map: VOICE (match-all)
    0 packets, 0 bytes
    5 minute offered rate 0 bps, drop rate 0 bps
    Match: access-group 110
    Queueing
      Strict Priority
      Output Queue: Conversation 264
      Bandwidth 25 (%)
```

LLQ is an excellent choice for voice traffic - after all, there's a special priority queue that's designed for voice traffic - but even with LLQ, you can run into some issues with voice delivery. If you do, you should look into running LFI.

And what is LFI? Glad you asked!

Link Fragmenting And Interleaving

- In today's networks, we've basically got two packet types: delay-sensitive, and non-delay-sensitive. Obviously, we want to get that delay-sensitive (voice, most likely) traffic to its destination as quickly as possible, but we can't just ignore the regular data traffic to do so.

LFI operates at Layer 2 and it allows these two traffic types to be sent almost simultaneously. It sounds complicated, but as we like to say around here, the name is the recipe.

The data packets will be *fragmented,* and then those fragments are *interleaved* with the delay-sensitive packets. (This is a fancy way of saying that the fragments are mixed in with the delay-sensitive packets as they're sent across the link.)

Once the fragments reach the other side of the link, they're put back together. Simple enough!

LFI is primarily used on Frame Relay and ATM circuits, although you can configure LFI on a dialer interface for use with ISDN. Interestingly enough,while LFI does support Voice Over IP (VoIP), it does not support Voice over Frame Relay (VoFM) or Voice over ATM (VoATM).

Let's walk through an example of configuring LFI on a frame circuit. First, we'll create a Virtual Template with our old friend ppp multilink. After we enable ppp multilink, we'll enable LFI.

```
R1(config)#interface virtual-template 5
R1(config-if)#ppp multilink
R1(config-if)#ppp multilink ?
  bap             Enable BACP/BAP bandwidth allocation negotiation
  fragment-delay  Specify the maximum delay for each fragment
  fragmentation   Enable/Disable multilink fragmentation
  idle-link       Do not transmit fragments over the lowest speed link
  interleave      Allow interleaving of small packets with fragments
  <cr>

R1(config-if)#ppp multilink interleave
```

To apply it to a given Frame Relay DLCI, enable traffic shaping on the interface and then apply the virtual template to the appropriate DLCI.

```
R1(config-if)#interface serial0

R1(config-if)#frame traffic-shaping
R1(config-if)#frame-relay interface-dlci 122 ?
  ppp  Use RFC1973 Encapsulation to support PPP over FR
  <cr>

R1(config-if)#frame-relay interface-dlci 122 ppp ?
  Virtual-Template  Virtual Template interface

R1(config-if)#frame-relay interface-dlci 122 ppp virtual-template ?
  <1-25>  Virtual-Template interface number
```

```
R1(config-if)#frame-relay interface-dlci 122 ppp virtual-template 5
```

There are other methods of configuring LMI, and the configuration for ATM is beyond the scope of the ONT exam. It is a good idea to know the basic configuration of LMI, and an especially good idea to know what it does!

"Hot Spots And Gotchas"

No matter which queueing solution you choose, there's a basic three-step process to follow - create the queues, divide traffic into classes, and schedule the queues in accordance with their importance.

You can only apply one queueing scheme to an interface.

- FIFO is fine in some cases, but it's *not* appropriate for voice, video, or high-priority application traffic in your network.

Low Latency Queueing (LLQ) is basically CBWFQ with a priority queue. That LLQ priority queue is designed for voice traffic. making LLQ the queueing scheme of choice over CBWFQ. CBWFQ is a better choice if the traffic is "regular", non-delay-sensitive data.

With Priority Queueing, queue starvation can occur if too many types of traffic are classified as High priority. Packets in the lower-importance queues can actually end up just sitting in the queue to allow the packets in the higher-importance queues to be transmitted.

CBWFQ uses ACLs; WFQ does not. Both of these allow us to define how much bandwidth should be assigned to each queue.

WFQ is the default for Serial interfaces running at E1 or less.

WFQ uses the concept of aggressive flows to decide which packets to drop when the queues become congested.

Copyright © 2008 The Bryant Advantage. All Rights Reserved.