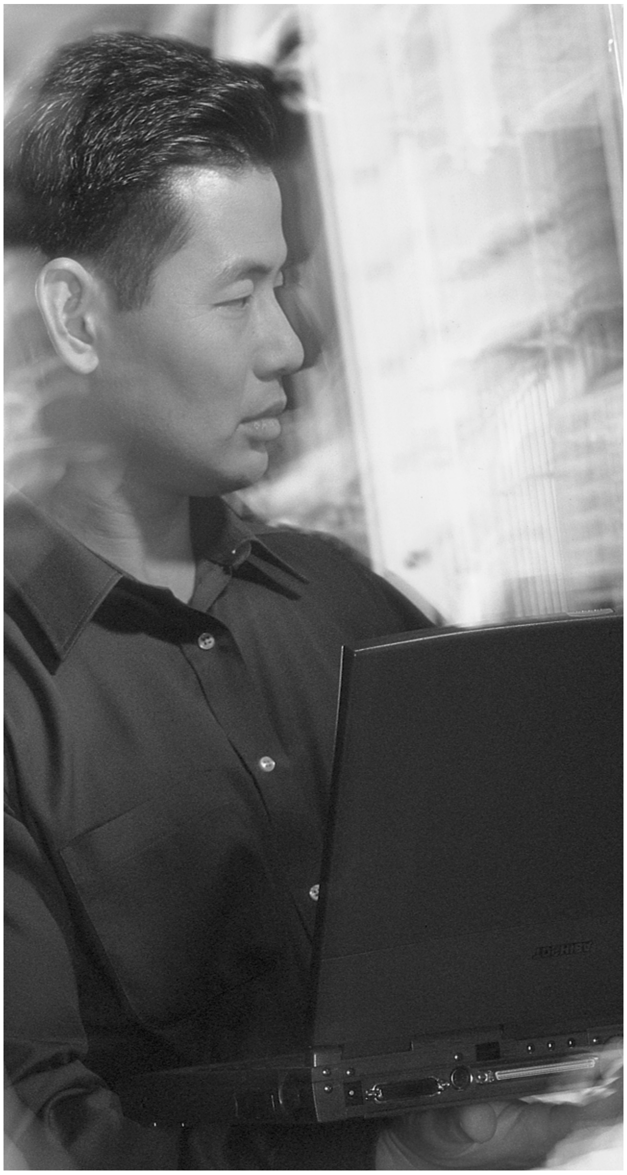


This chapter covers the following subjects:

- Queuing Overview
- Introduction to Queuing
- Class-Based Weighted Fair Queuing (CBWFQ)
- Low-Latency Queuing (LLQ)
- Compression Techniques

CHAPTER 15

Managing Network Performance with Queuing and Compression

The CCNP Remote Access exam requires you to have an in-depth understanding of various WAN technologies. In this chapter, the discussion focuses on some of the advanced queuing techniques offered in Cisco IOS. The latter part of this chapter also discusses compression techniques that are possible from Cisco routers.

There are various opinions on whether queuing is needed in a router, or in a network. Normally, there are ways to justify the need for queuing, and the type of queuing required. This chapter discusses the concepts behind queuing, when to use queuing, and which type of queuing is best for a particular situation.

"Do I Know This Already?" Quiz

The purpose of the "Do I Know This Already?" quiz is to help you decide whether you really need to read the entire chapter. If you already intend to read the entire chapter, you do not necessarily need to answer these questions now.

This ten-question quiz, derived from the major sections in the "Foundation Topics" portion of the chapter, helps you determine how to spend your limited study time.

Table 15-1 outlines the major topics discussed in this chapter and the "Do I Know This Already?" quiz questions that correspond to those topics.

Table 15-1 *"Do I Know This Already?" Foundation Topics Section-to-Question Mapping*

Foundation Topics Section	Questions Covered in This Section
Queuing Overview	1–2
Introduction to Queuing	3–4
Class-Based Weighted Fair Queuing	5–6
Low-Latency Queuing	7–8
Compression Techniques	9–10

CAUTION The goal of self-assessment is to gauge your mastery of the topics in this chapter. If you do not know the answer to a question or are only partially sure of the answer, you should mark this question wrong for purposes of the self-assessment. Giving yourself credit for an answer you correctly guess skews your self-assessment results and might provide you with a false sense of security.

1. Which network conditions justify the use of queuing? (Select two.)

- **a.** High-bandwidth interfaces
- **b.** Congested interfaces
- **c.** Low-speed interfaces
- **d.** LAN interfaces
- **e.** Underutilized interfaces

2. Which effects does queuing have on a router? (Select two.)

- **a.** Increases router performance
- **b.** Decreases CPU load
- **c.** Increases CPU load
- **d.** Increases the bandwidth on an interface
- **e.** Decreases router performance

3. Which queuing strategy uses the IP type of service (ToS) bits to help determine egress priority?

- **a.** First-In, First-Out queuing
- **b.** Fair Queuing
- **c.** Weighted Fair Queuing
- **d.** Priority Queuing

4. How are packets sequenced when using FIFO queuing?

- **a.** The shortest packets always go to the front of the queue.
- **b.** The longest packets always go to the front of the queue.
- **c.** The packets are sequenced based on when each respective first bit arrives on the egress interface.
- **d.** The packets are sequenced based on when each packet arrives entirely on the egress interface.

5. Which Cisco IOS command is used to create a list of packets that match one or more criteria?

- **a.** **class-map**
- **b.** **policy-map**
- **c.** **service-policy**
- **d.** **class-list**

6. Which Cisco IOS command determines how much bandwidth a particular flow should get during congested conditions?

- **a.** **class-map**
- **b.** **bandwidth**
- **c.** **service-policy**
- **d.** **policy**

7. Which Cisco IOS command determines how much bandwidth a particular flow should get at all times?

- **a.** **class-map**
- **b.** **bandwidth**
- **c.** **service-policy**
- **d.** **policy**

8. What is the difference between CBWFQ and LLQ?

- **a.** There is no difference because they use the same IOS commands.
- **b.** CBWFQ defines a priority queue that is used at all times whereas LLQ is used only during congestion.
- **c.** LLQ defines a priority queue that is used at all times whereas CBWFQ is used only during congestion.
- **d.** CBWFQ has more configuration options than LLQ.

9. Which of the following compression methods can be used only on point-to-point links? (Select two.)

- **a.** TCP header compression
- **b.** Stac
- **c.** Predictor
- **d.** MPCC
- **e.** IP header compression

10. Which of the following compression methods is Cisco-proprietary?

  a. TCP header compression
  b. Stac
  c. Predictor
  d. MPCC

The answers to the "Do I Know This Already?" quiz are found in Appendix A, "Answers to the 'Do I Know This Already?' Quizzes and Q&A." The suggested choices for your next step are as follows:

- **6 or fewer overall score**—Read the chapter. This includes the "Foundation Topics," "Foundation Summary," and the "Q&A" section.
- **7 or 8 overall score**—Begin with the "Foundation Summary" section and then go to the "Q&A" section.
- **9 or more overall score**—If you want more review of these topics, skip to the "Foundation Summary" section and then go to the "Q&A" section. Otherwise, move to the next chapter.

Foundation Topics

Queuing Overview

Queuing is the process of sequencing packets before they leave a router interface. Normally, packets leave the router in the order they arrived. This first-in, first-out (FIFO) process does not give any special attention to voice or mission-critical traffic through the router. Today's networks may require special sequencing to ensure that important packets get through the router in a timely fashion. Cisco routers have a variety of queuing techniques that may be used to improve traffic throughput.

Along with a discussion of traffic queuing, this chapter also addresses the topic of compression. Compression is a somewhat misunderstood tool. Although compression has a number of situations in which it is useful, it has just as many circumstances in which it is detrimental.

The misconception that queuing is a necessary part of any router configuration is a topic that needs to be dealt with up front. As mentioned, any queuing strategy results in higher delay in the network, because of a higher per-packet processor requirement. In other words, each traffic type must be sorted out and dealt with according to the defined parameters of the queue. This is the trade-off for assuring that your critical traffic passes through the router. In an oversimplified view, queuing is simply sorting packets into some new sequence before they leave any particular interface.

Queuing is only necessary when the existing traffic flow is having problems getting through the router. If all traffic is going through properly and no packet drops are occurring, leave it alone. Simply put, in the absence of congestion, do not implement a queuing strategy and leave the default setting alone. Depending on the interface type and speed, a queuing strategy might already be in place. Again, if it works, do not change it. That point cannot be stressed enough.

It is also important to remember that regardless of the queuing method that is employed, in most cases, it never actually functions unless the egress interface is busy. If an interface is not stressed, and there is no outbound packet congestion, then the queuing strategy is not applied (LLQ is the exception to this rule). It you want the queuing policy to work at all times, you must adjust the outbound hardware buffers to make it appear that the interface is always busy.

The concept of queuing is shown in Figure 15-1. In this diagram, packets are arriving at an interface at various intervals. Some are considered more important than others, but the router is not capable of distinguishing this by itself. Also, the flow of packets toward the interface is greater than the interface can empty out on the other side.

Figure 15-1 *Queuing Concepts*

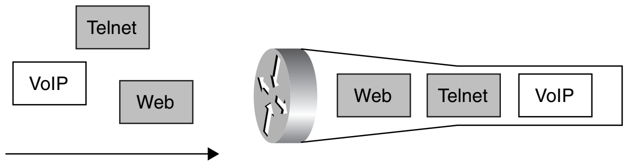


Figure 15-1 shows that the voice packet arrived last, yet exits first. Also, small, interactive packets such as Telnet packets have the next highest priority. And finally, the generic web packets are sent through the router. Without a defined queuing mechanism, packets would be sent FIFO. In Figure 15-1, FIFO would not be a good option, because the voice and Telnet packets would suffer.

There are two advanced types of queuing discussed in detail later in this chapter:

- Class-Based Weighted Fair Queuing (CBWFQ)
- Low-Latency Queuing (LLQ)

Before CBWFQ is explored, the concepts leading up to it need to be reviewed. Thus, First-In, First-Out (FIFO queuing, Fair Queuing (FQ), and Weighted Fair Queuing (WFQ) are briefly covered. This background information makes it much easier to understand CBWFQ. LLQ is a natural follow-on to, and actually an extension of, CBWFQ.

Queuing is most effectively implemented on WAN links. Bursty traffic and low data rates can combine to create a congestive situation that can require administrative oversight to correct. Depending on the maximum transmission units (MTUs) of the surrounding media, queuing is most effective when applied to links with T1 (1.544 Mbps) or E1 (2.048 Mbps) bandwidth speeds or lower. In fact, any serial interfaces on a Cisco router use WFQ by default if the throughput (clockrate) is less than or equal to 2 Mbps.

If congestion is temporary, queuing can be a proper remedy to the situation. If the congestion is chronic, queuing can compound the issue by introducing additional delay. If congestion is a constant issue, then it might be time to accept the fact that a bandwidth upgrade (and possibly a router upgrade) is in order. Although a circuit or hardware upgrade will cost considerably more than implementing a policy change, there are times when there is no other choice.

The establishment of a queuing policy assists the network administrator with handling individual traffic types. The goal, typically, is to maintain the stability of the overall network, even in the face of numerous traffic needs and types. Unfortunately, a lot of time can be spent supporting traffic types that are not in line with company goals. Some administrators will transport all traffic, regardless of whether it is really necessary. Sometimes, it might be difficult to create or enforce a policy of policing traffic (throwing out stuff that does not belong). Thus, queuing is necessary to sequence the important traffic first, and maybe leave the less important stuff to the back of the line.

Queuing is an organization policy. It decides the order that packets leave any given interface. Queuing does not increase bandwidth. It simply works within the parameters of an interface and best utilizes those parameters. Note that different queuing strategies have different opinions on the term "best."

Once the decision has been made to implement a queuing strategy, you must decide which queuing strategy should be utilized. Figure 15-1 serves as a fundamental map to assist in that decision. As shown in the figure, you must determine whether the level of congestion constitutes a condition that requires queuing. Once you make that determination, another decision awaits. How strictly should the control of the queuing policy be enforced? Are the defaults OK, or should a more granular approach be applied?

Introduction to Queuing

Cisco routers support a wide variety of queuing methodologies. Some have been around quite some time. Others are more modern, and can be quite complex and difficult to understand. Due to the complexity of more modern queuing technologies, it is difficult to describe and understand them without first understanding the queuing basics that lead up to the more complex methods.

This section describes the following three queuing methods that lead up to the complex queuing methods discussed later:

- First-In, First-Out (FIFO) Queuing
- Fair Queuing (FQ)
- Weighted Fair Queuing (WFQ)

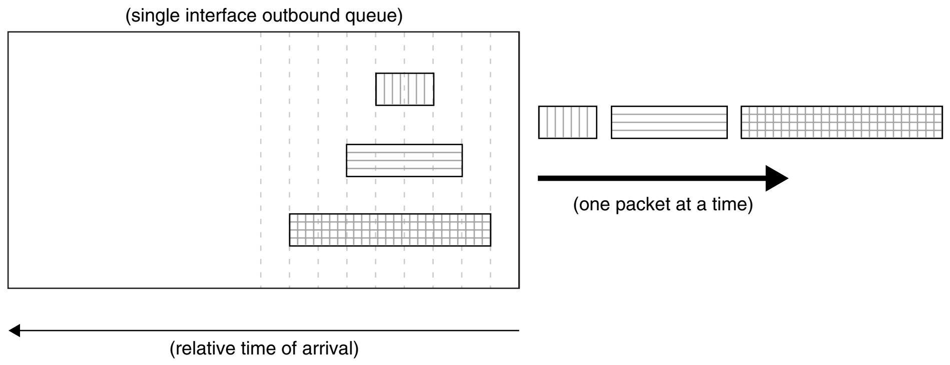
First-In, First-Out Queuing

FIFO queuing is the most basic of strategies. In essence, it is the first-come, first-served approach to data forwarding. In FIFO, packets are transmitted in the order in which they are received. Keep in mind that this process occurs on each interface in a router, not in the router as a whole.

On high-speed interfaces (greater that 2 Mbps), FIFO is the default queuing strategy on a router. Normally, such high-bandwidth interfaces do not have problems getting traffic out the door.

Figure 15-2 displays the basic model of FIFO. Notice that there are three different sizes of packets. One potential problem of FIFO is that the small packets must wait in line for the larger packets to get dispatched. In the figure, the smallest packet is actually ready to leave before the largest packet is finished arriving. However, because the largest packet started to arrive at the interface first, it gets to leave the interface first. This actually causes gaps between data on the wire, which decreases efficiency.

Figure 15-2 *FIFO*



FIFO is not really queuing; it is more along the lines of buffering. The packets are routed to the interface and stored (queued) in router memory until transmittal. The transmission order is based on the arrival order of the first bit of the packet, even though the last bit may be still far away. Essentially, the outbound packet buffer is selected as soon as its outbound interface is selected.

Fair Queuing

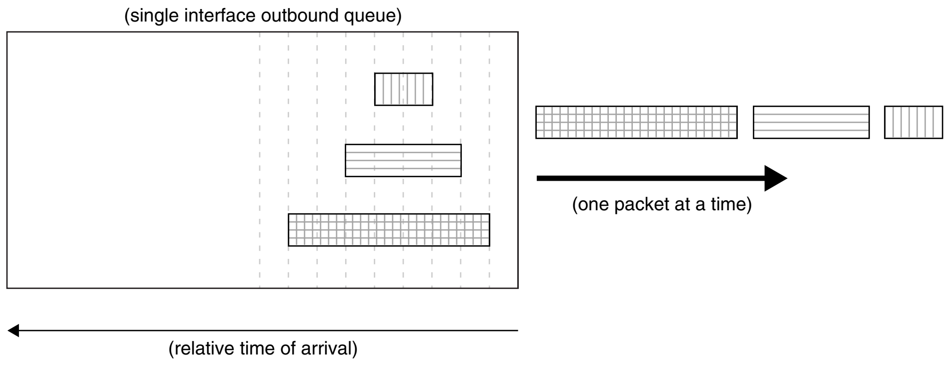
FIFO queuing does not offer any way to permit packets that are "ready" to be transmitted to leave before packets that are still "preparing" to be transmitted. As was demonstrated in Figure 15-2, large packets, based on arrival time, can clog an outbound interface because their first bit was first to arrive on the interface.

Fair Queuing is a methodology that allows packets that are ready to be transmitted to leave, even if they started to arrive after another packet. Note that FQ is not an option in Cisco routers, but understanding FQ will help you to understand WFQ.

Using the same example as before, the effects of FQ are shown in Figure 15-3. The same data flow is sent to the egress interface, only this time the smallest packets are allowed to leave first because they are ready to leave before the larger packet.

FQ allows smaller packets to "cut the line" in front of larger packets that are still in the process of arriving. This process solves the FIFO problem of gaps between packets on the wire caused by the blocking by the large packets.

Figure 15-3 *Fair Queuing*



Weighted Fair Queuing

As mentioned, FIFO is often not the ideal queuing method on low-bandwidth interfaces. Different data patterns may suffer in a FIFO environment. Consider Telnet and FTP competing for the same egress interface. With FIFO, the small Telnet packets must wait behind the large FTP packets. With FQ, the small Telnet packets are allowed to leave once each packet has completely arrived at the interface. When a large FTP packet is ready to go, it is dispatched. Then, while another large FTP packet is building in the buffer, multiple Telnet packets are sent.

However, FQ does not take into account any parameters stored within the packet, such as type of service (ToS). Some small packets, such as voice, should have a higher priority than other small packets, such as Telnet. If size were the only delimiter (FQ), then small voice packets would be considered the "same" as small Telnet packets. This could cause delay and jitter in the voice quality.

WFQ starts by sorting traffic that arrives on an egress interface into conversations or flows. The router determines what the actual flows are, and the administrator cannot influence this decision. Basically, the conversations are based on a hash (combination) of the source and destination IP addresses, protocol number, ports, MAC addresses, DLCI numbers, etc. Not all values may be used to determine any flow. The ToS is not used to determine flow.

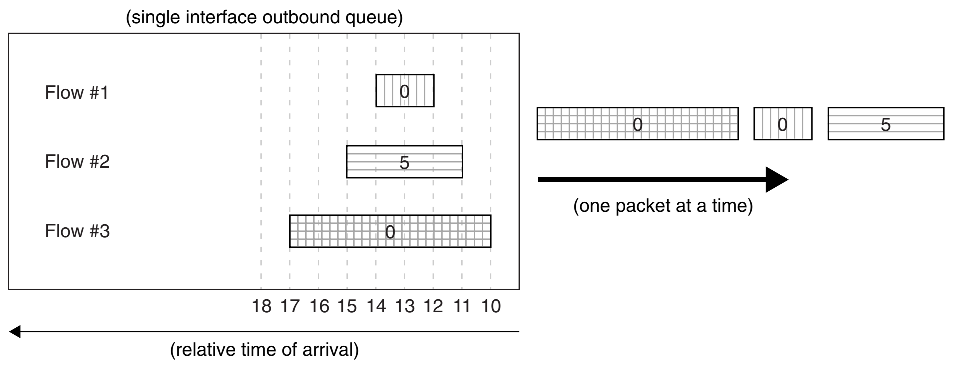
The administrator can define the maximum number of flows possible. The router performs the flow selection. WFQ dispatches packets from the front of any given flow only. Thus, a packet in the middle of flow #2 cannot be dispatched until all the packets at the front of flow #2 are sent. In other words, each flow is handled in FIFO order.

WFQ differs from FQ because it uses the ToS bits that travel within each IP header. Remember that FQ looks at when a packet finished arriving (relative time) to determine when it actually is dispatched. Thus, the priority of the packet specified in the ToS bits becomes a "weight" when dispatching packets through an egress interface.

WFQ multiplies this relative time by a mathematical variation of the ToS to determine the new "when to dispatch" number. For this description, and to simplify the math, ToS 7 = multiplier 1, ToS 6 = multiplier 2, down through ToS 0 = multiplier 8. In reality, the multiplier numbers are much larger, but on a similar scale.

Figure 15-4 shows how the WFQ system works. Three packets have arrived on this egress interface. The router, configured for FQ on this interface, has determined that there are three different flows. The administrator cannot impact the flow selection process. The relative arrival time is shown below the queue.

Figure 15-4 *Weighted Fair Queuing*



For FIFO, the largest packet would be dispatched first, followed by the medium one, followed by the smallest. FQ corrects this by sending the smallest first, then the medium one, then the largest. But in this new example, the medium packet has a much higher priority (ToS = 5) than the small packet (ToS = 0). Thus, WFQ adjusts the dispatch accordingly.

Remember that all values shown here for the "multiplier" are adjusted for simple mathematical examples. Real numbers are much larger, but on a similar scale.

The large packet starts arriving at time 10, but finishes at time 17. With a ToS of 0, the multiplication factor is 8. Thus, $17 \times 8 = 136$. The medium packet starts arriving at time 11, but finishes at time 15. Its ToS of 5 has a multiplication factor of 3. Thus $15 \times 3 = 45$. And finally, the small packet starts arriving at time 12, however it finishes at time 14. ToS 0 = multiplier 8, thus $14 \times 8 = 112$.

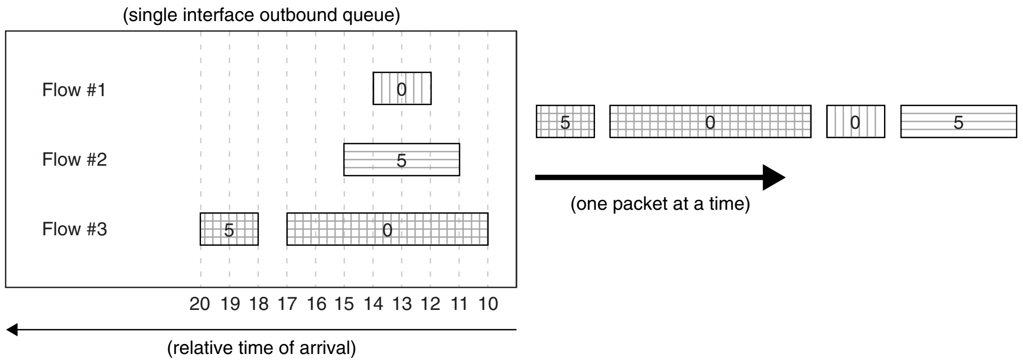
Table 15-2 takes all these potentially confusing numbers and arranges them logically.

Table 15-2 *Weighted Fair Queuing*

Packet	Start	Finish	ToS	Multiplier (based on ToS)	Dispatch (finish × multiplier)
Small	12	14	0	8	112
Medium	11	15	5	3	45
Large	10	17	0	8	136

So, the medium packet is dispatched first, followed by the small packet, followed by the large one. So it seems that WFQ solves the problems of getting small packets out first and ensuring that higher-priority packets get fair usage of the bandwidth. However, because the administrator cannot control the selection of the conversations (or flows), WFQ does have a few issues. Consider Figure 15-5.

Figure 15-5 *WFQ #2*



In this example, the third flow has two packets. However, the second packet is a high-priority packet (ToS = 5). It is quite possible to have packets of various ToS in a single flow. Remember that dynamic flow selection is not based on ToS.

The problem here is that the high-priority packet in flow #3 cannot be dispatched until after the large packet in front of it (same flow) leaves. Packets within a flow are handled FIFO. The WFQ algorithm only works with the first packets in each of the dynamically created flows. And as mentioned, the administrator has no control over how packets get sorted into the flows.

Thus, in the scenario shown, although it would be nice (and probably desired) to have the high-priority packets leave first, it is not the case. The high-priority packet in flow #3 is actually the last one out the door.

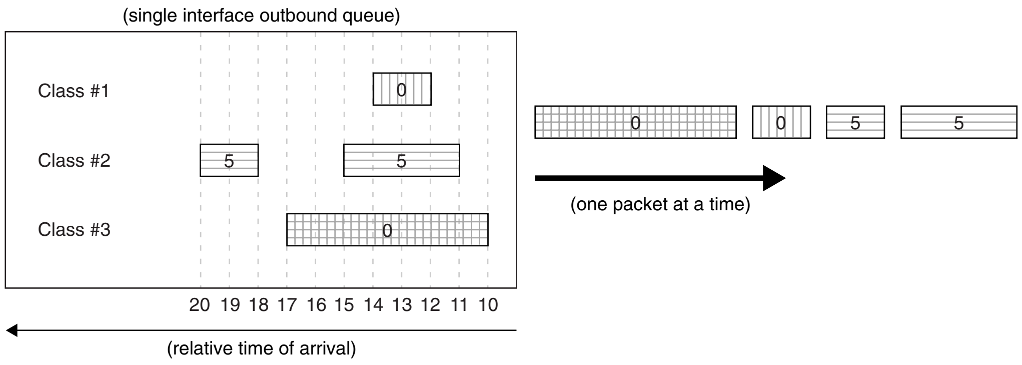
Class-Based Weighted Fair Queuing

Because all the problems associated with getting the proper packets out first have not been addressed (thus far), additional queuing methods have been developed. The basic building block of WFQ offers a nice starting point for additional queuing strategies. WFQ separates the packets into flows and applies a weight to high-priority packets so that they can leave first, but as the previous section shows, the potential shortcoming of WFQ is the lack of administrator control.

CBWFQ adds a level of administrator control to the WFQ process. The same WFQ dispatch process is followed. The difference now is that the administrator can control how packets are divided into the conversations or flows.

Thus far, FIFO and WFQ really do not need much configuration. FIFO is really the lack of WFQ, and WFQ is on by default for interfaces that are less than 2 Mbps. There are a few commands to tweak WFQ, but there are no commands to control WFQ. CBWFQ, on the other hand, can be controlled. Figure 15-6 shows how CBWFQ tames WFQ.

Figure 15-6 *Class-Based Weighted Fair Queuing*



In this example, the administrator has decided that all high-priority traffic should reside in the same flow, regardless of any other conditions that might place them into separate flows. The administrator-defined flows are called classes. The WFQ algorithm is still at work, but the queue definition is under control now. Table 15-3 shows that although the second high-priority packet arrived well after the large packet, it still goes out second in line.

CBWFQ allows the administrator to define the various flows needed to classify traffic into unique conversations. In addition to this separation of traffic, CBWFQ can be used to guarantee that flows receive adequate bandwidth (defined by the administrator).

Table 15-3 *CBWFQ Uses WFQ*

Packet	**Start**	**Finish**	**ToS**	**Multiplier (based on ToS)**	**Dispatch (finish × multiplier)**
Small #1	12	14	0	8	112
Medium	11	15	5	3	45
Large	10	17	0	8	136
Small #2	18	20	5	3	60

CBWFQ has three basic components. The first part is defining the flows for specific traffic patterns (**class-map** command). There are many different ways to determine such flows. The second part is deciding how to handle each flow (**policy-map** command). Normally, minimum and/or maximum bandwidth requirements are set on each flow. And the final part is to assign this policy to the interface (**service-policy** command). CBWFQ can also be assigned to a Frame Relay DLCI. The following three sections describe each of these components in turn. The actual configuration and verification commands are then described.

The class-map Command

The first configuration part of CBWFQ is to define the individual flows that are to pass through the queuing system. The **class-map** command, which is used to create these flows, selects specific traffic to be put into the CBWFQ. It is possible to have many **class-map** commands, and each one can have up to 64 matching conditions within. Example 15-1 shows how to create a class map.

Example 15-1 *The* **class-map** *Command*

```
class-map match-all bcran-class
```

The **class-map** command, as shown in Example 15-1, defines a named class (using *bcran-class* as a name) that is used to select traffic. There are two options used when a class map is defined. The preceding command uses the **match-all** option. This means that all **match** commands within the class map must be true for this class to succeed. Thus, this is a logical AND condition. The alternate is the **match-any** command. This states that any one of the **match** commands causes this class to succeed. This is the logical OR condition.

Within the class map, **match** commands are used to find packets for this class. Example 15-2 shows a class map with a single **match** command.

Example 15-2 *The* **match** *Command*

```
class-map match-all bcran-class
  match access-group 101
```

In Example 15-2, any packets that are permitted in IP access list 101 are put into the class map *bcran-class*. Because only a single match statement is used, the class map could be either **match-all** or **match-any**.

In Example 15-3, only packets that are permitted in both IP access lists 101 and 102 are entered into the class map.

Example 15-3 *The* **match-all** *Command*

```
class-map match-all bcran-class
  match access-group 101
  match access-group 102
```

In Example 15-4, packets that are permitted by either IP access list 101 or 102 are put into the class map. The difference this time is the use of the **match-any** command when defining the class map.

Example 15-4 *The* **match-any** *Command*

```
class-map match-any bcran-class
  match access-group 101
  match access-group 102
```

The **match** command can be used to examine a wide variety of different criteria. As has been shown, IP traffic in access lists can be examined. Also, the IP precedence value, IP DSCP value, IP RTP ports, COS bits, QoS group number, MPLS experimental bits, and protocol values can be matched in a **class-map**.

The policy-map Command

A class map consists of one or more **match** commands to select packets for the class. A **policy-map** collects one or more **class-maps** and defines the actions that are taken for each **class-map**. Thus, for a **policy-map** command to properly function, a **class-map** must already exist. Example 15-5 shows how a **policy-map** is created.

Example 15-5 *The* **policy-map** *Command*

```
policy-map bcran-policy
  class bcran-class
    bandwidth 48
```

Example 15-5 first defines a **policy-map** named *bcran-policy*. The **policy-map** then maps the **class-map** named *bcran-class* (created earlier). And, all packets that are officially a member of the **class-map** are given a minimum bandwidth of 48 kbps. The bandwidth option states that the **class-map** is guaranteed 48 kbps. Note that the guarantee of bandwidth is not enforced unless the interface is congested.

In reality, the **class-map** may use more bandwidth if it is available, but if the interface where this policy is applied is busy, this particular **class-map** will get 48 kbps.

Example 15-6 shows the same **policy-map** named *bcran-policy*, but this time, there are three different **class** statements used.

Example 15-6 *A Complete* **policy map** *Configuration*

```
policy-map bcran-policy
  class bcran-class
    bandwidth 48
  class other-class
    bandwidth 24
  class class-default
    fair-queue
```

Two of the **class** statements reference specific class maps that have already been defined. The first class, called *bcran-class*, gets a minimum of 48 kbps. The second one, called *other-class*, gets a minimum of 24 kbps. And everything else that passes through the interface where this policy is applied uses WFQ. The function of the *class-default* is to catch any traffic that does not match any of the class maps within the policy map.

The preceding configuration example allocates a specific amount of bandwidth to each **class-map**. Bandwidth can also be allocated to a percentage of the total available bandwidth on an interface, or as a percentage of the remaining bandwidth not claimed by any other **class-maps**. Other configuration options include policing traffic (throwing out) to either a percentage of bandwidth or to a specific bandwidth value; shaping traffic (buffering); setting various markers; and adjusting the queue limits to avoid tail drops.

The service-policy Command

Now that the CBWFQ policy has been constructed, it must be applied to an interface. The **service-policy** command is used to map an existing policy map to an interface. Note that CBWFQ policies can be applied to either incoming or outgoing traffic flows.

Example 15-7 shows how to apply a prebuilt **policy-class** to an interface.

Example 15-7 *The* **service-policy** *Command*

```
interface serial 0/0
  service-policy output bcran-policy
```

Only one **policy-class** can be applied to an interface in one direction. Normally, the policy is applied for egress (output) traffic, because the low-bandwidth region is outside the interface.

The completed configuration that has been discussed piece-by-piece throughout this chapter thus far is shown in Example 15-8.

Example 15-8 *The Complete CBWFQ Configuration*

```
! ACL 101 permits telnet from 172.16.1.0/24 to 192.168.1.0/24
access-list 101 permit tcp 172.16.1.0 0.0.0.255 192.168.1.0 0.0.0.255 eq telnet

! ACL 102 permits priority 5 traffic from 172.16.2.0/24 to 192.168.2.0/24
access-list 102 permit ip 172.16.2.0 0.0.0.255 192.168.2.0 0.0.0.255 precedence critical

! ACL 103 permits priority 5 traffic from 172.16.3.0/24 to 192.168.3.0/24
access-list 103 permit ip 172.16.3.0 0.0.0.255 192.168.3.0 0.0.0.255 precedence critical

! class-map bcran-class1 matches anything from either ACL 101 or 102
class-map match-any bcran-class
  match access-group 101
  match access-group 102

! class-map bcran-class2 matches anything from ACL 103
class-map match-any other-class
  match access-group 103

! policy-map bcran-policy allows class-map bcran-class 48Kbps,
! class-map other-class 24Kbps, and all other traffic is WFQ
policy-map bcran-policy
  class bcran-class
    bandwidth 48
  class other-class
    bandwidth 24
  class class-default
    fair-queue

! policy-map bcran-policy is applied to outbound traffic on serial 0/0
interface serial 0/0
  service-policy output bcran-policy
```

CBWFQ Verification

Once CBWFQ has been configured, the individual pieces can be examined from the command-line interface (CLI). The first thing that should be examined is the interface queuing policy. CBWFQ is an upgrade to WFQ. Thus, the queue examination between WFQ and CBWFQ are quite similar. Example 15-9 shows an interface configured for WFQ.

Example 15-9 *The* **show queue** *Command*

```
Router# show queue serial 0/0
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: weighted fair
  Output queue: 0/1000/64/0 (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
! Next, the same interface is shown when the CBWFQ policy is applied:
Router# show queue serial 0/0
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: weighted fair
Output queue: 0/1000/64/0 (size/max total/threshold/drops)
     Conversations  0/1/256 (active/max active/max total)
     Reserved Conversations 2/2 (allocated/max allocated)
     Available Bandwidth 1086 kilobits/sec
```

At first glance, it may appear that these two queue displays are identical. Both claim that the interface is configured for WFQ. However, closer examination shows that the second display has two reserved conversations. These reservations represent the two **class-map** references from the policy map that is applied to this interface. However, there is no specific reference to which policy map may be applied to this interface. Thus, the actual **policy-map** must be displayed to see the actual **class-maps** within.

In Example 15-10, the **policy-map** shows that there are three class statements.

Example 15-10 *The* **show policy-map** *Command*

```
Router# show policy-map
  Policy Map bcran-policy
    Class other-class
      Weighted Fair Queueing
            Bandwidth 24 (kbps) Max Threshold 64 (packets)
    Class bcran-class
      Weighted Fair Queueing
            Bandwidth 48 (kbps) Max Threshold 64 (packets)
    Class class-default
      Weighted Fair Queueing
            Flow based Fair Queueing
            Bandwidth 0 (kbps) Max Threshold 64 (packets)
```

In this example, *other-class* allocates 24 kbps to packets, *bcran-class* gives 48 kbps to packets, and the *class-default* class does not promise any bandwidth to the remaining packets. All other packets are handled via WFQ. However, packets that match any of the displayed class maps are only known by examining the **class-maps**.

In Example 15-11, the contents of both class maps are shown.

Example 15-11 *The* **show class-map** *Command*

```
Router# show class-map
 Class Map match-any other-class (id 3)
   Match access-group  103

 Class Map match-any class-default (id 0)
   Match any

 Class Map match-any bcran-class (id 2)
   Match access-group  101
   Match access-group  102
```

class-map *other-class* has a single **match** statement and looks at only ACL 103. **class-map** *bcran-class* has two **match** statements for two different ACLs. The ACLs are "OR'd" together, due to the **match-any** statement. The *class-default* simply matches everything that has not been matched thus far.

The access lists that are actually the basis for the matching are shown in Example 15-12.

Example 15-12 *The* **show access-list** *Command*

```
Router# show access-lists
Extended IP access list 101
    permit tcp 172.16.1.0 0.0.0.255 192.168.1.0 0.0.0.255 eq telnet
Extended IP access list 102
    permit ip 172.16.2.0 0.0.0.255 192.168.2.0 0.0.0.255 precedence critical
Extended IP access list 103
    permit ip 172.16.3.0 0.0.0.255 192.168.3.0 0.0.0.255 precedence critical
```

As previously shown in Example 15-8, ACL 101 permits a particular Telnet session. ACL 102 permits a particular series of packets with the priority set to 5. And ACL 103 also deals with priority 5 packets, but this time for a different packet stream.

Low-Latency Queuing

Low-Latency Queuing (LLQ) is really just an extension of CBWFQ. In fact, the only real difference between the two is how the bandwidth is allocated to the class maps in the policy map.

In the examples shown thus far, the **bandwidth** command was used to allocate a certain amount of bandwidth to any given class map. Remember that the **bandwidth** command has an effect in the queuing strategy only if the interface is congested. Thus, if the interface is not filled up, there is no real guarantee that any particular class map will get the requested amount of bandwidth.

LLQ uses the **priority** command instead of the **bandwidth** command to request bandwidth. The **priority** command guarantees that the requested bandwidth is available whether the interface is busy or not. Because this bandwidth is always available, the class map that uses the **priority** command is guaranteed low latency through the interface (thus the name, LLQ). This is also called a strict priority queue.

It is important to remember that any packets that exceed the requested bandwidth (using the **priority** command) when the interface is busy are discarded (policed). During low interface utilization, the class map may use more bandwidth than requested by the **priority** command. But the class map will get the required bandwidth at all times.

The policy-map Command

LLQ is configured the same as CBWFQ. The difference is how the bandwidth is requested in the policy map. Example 15-13 shows the use of the **priority** command.

Example 15-13 *The* **policy-map** *Command*

```
policy-map bcran-policy
  class bcran-class
    priority 48
```

The difference here is that **class-map** *bcran-class* is absolutely guaranteed 48 kbps of bandwidth at all times, regardless of how busy the interface is.

LLQ Verification

Because LLQ is nearly identical to CBWFQ, most of the verification screens are the same. The one place where the difference can be seen is when the policy map is examined. Example 15-14 examines the **policy-map** built for LLC.

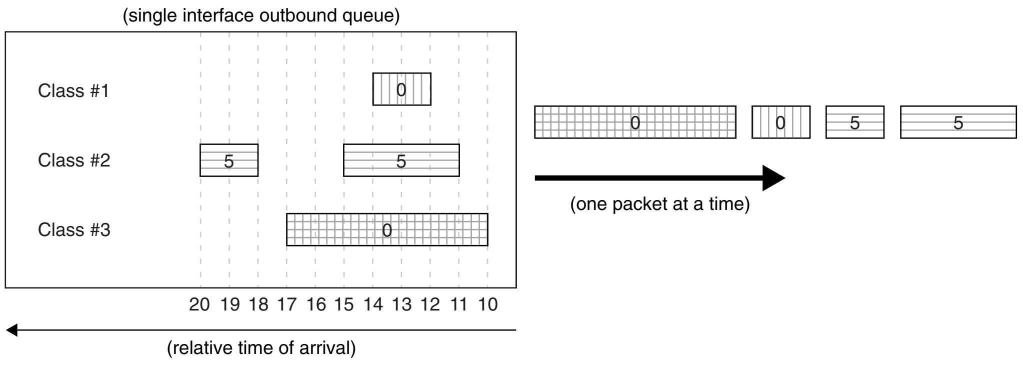
This time, **class-map** *bcran-class* is shown with a strict priority of 48 kbps. This verifies that it is guaranteed the requested bandwidth under all circumstances.

Example 15-14 *The* **show policy-map** *Command*

```
Router# show policy-map
  Policy Map bcran-policy
    Class other-class
      Weighted Fair Queueing
            Bandwidth 24 (kbps) Max Threshold 64 (packets)
    Class bcran-class
      Weighted Fair Queueing
            Strict Priority
            Bandwidth 48 (kbps) Burst 1200 (Bytes)
    Class class-default
      Weighted Fair Queueing
            Flow based Fair Queueing
            Bandwidth 0 (kbps) Max Threshold 64 (packets)
```

As shown in Figure 15-7, the flow of packets through the LLQ is almost identical to that of the flow through the CBWFQ. The only real difference is that the priority traffic in the LLQ is always guaranteed its prescribed bandwidth. The CBWFQ is only in operation when the interface is congested.

Figure 15-7 *Low-Latency Queuing*



Compression Techniques

Various types of compression algorithms are in use in the world today.

For compression, a scope needs to be set ahead of time. There are compression methods for data (entire files), links (data that travels between routers), hard drives (data stored on a hard drive), and so on. This section of the chapter focuses on compression across WAN links (links).

It is also true that too much compression is a bad thing. If data is already compressed when WAN links begin to process it, the ability of the router to further compress that data is affected. Data that is already compressed can actually become larger by recompressing it. The discussion in this chapter focuses on what happens at the WAN interface, regardless of the type of data being transported.

Compression is only one technique for squeezing every possible bit of bandwidth from an existing internetwork deployment. Compression, like queuing, is meant to provide critical time to plan and deploy network upgrades and to reduce overall utilization of a WAN link. However, nothing is free. The execution of the compression algorithm adds a significant number of CPU cycles. Unfortunately, the additional load on the CPU might not be something it can handle.

With compression enabled, CPU utilization of the router increases considerably. On the bright side, the WAN link utilization drops considerably. Thus, compression is a trade-off, because all that has been accomplished is the displacement of utilization from the WAN to the router. Obviously, the effects of compression vary based on the algorithm implemented.

As technology advances, compression will move from a software function to a hardware function. This is already a reality in some router models, with the addition of newly available modules specifically geared toward performing data compression in hardware. Not only is this much faster than software compression, it is less costly for the CPU (generally).

The effects of compression must be taken into account prior to any implementation. If the routers are already running at 80 percent or more CPU utilization (**show cpu process** command), it is not a good idea to implement compression. Doing so can result in the router literally running out of CPU to do any further processing.

Data compression makes efficient use of bandwidth and increases WAN throughput by reducing the size of the frame being transported. Compression is best utilized on slower WAN links. There comes a point when the router can send the data faster if it is uncompressed than if the router compresses the data, sends it, and then decompresses the data at the other end.

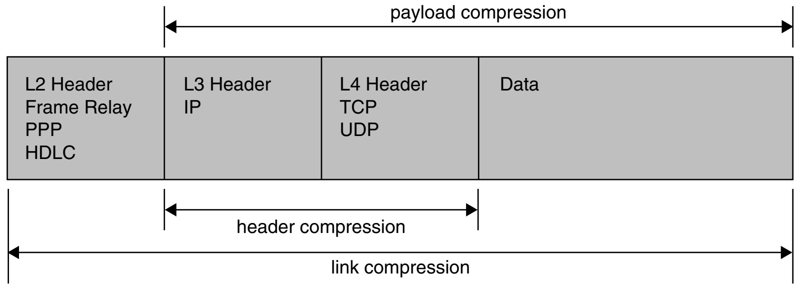
Cisco supports a number of compression types:

- Link
- Payload
- TCP Header
- Microsoft Point-to-Point

Microsoft Point-to-Point compression, which is an algorithm, is beyond the scope of this book and thus is not discussed further.

Figure 15-8 illustrates where various types of compression make transmission more efficient.

Figure 15-8 *Compression Methods*



Link Compression

Link compression (also known as *per-interface compression*) compresses the entire frame (Layer 2 through Layer 7) as it leaves the interface. In other words, both the header and the data are compressed. Because the entire frame is compressed, it must be uncompressed immediately upon receipt at the other end of the link so that the header information can be used to further forward the frame. Thus, a link-compressed frame is only compressed for the trip across one link.

Link compression is not dependent on any particular protocol function. Every frame that leaves the interface that is configured for compression is reduced in size—no questions asked, no exceptions to the rule. Cisco supports two algorithms on its router chassis to compress traffic: Stac and Predictor. For HDLC links, Stac is the only available choice.

Because the header and data are unusable after being compressed, link compression should be used only for data transmission over point-to-point dedicated connections.

It is also important to remember that if compression is used on one side of a link, compression must also be enabled on the other side, and the same compression algorithm must be used at both ends. Compression can be compared to a language. If English is spoken on one end, then English must be present on the other end to decipher the communication.

Stac

Stac is based on an algorithm known as Lempel-Ziv (LZ), which searches the data stream for redundant strings and replaces them with a token. The token is an information pointer that is significantly shorter than the string it replaces. If LZ cannot find any duplicated strings in the data, no compression occurs and transmission occurs as if the link had no compression activated. Stac is an open-standard compression algorithm and is used by many different vendors.

There are cases, such as the sending of encrypted data or data that has already been compressed, in which compression actually expands the size of a transmission. In such cases, the original transmission is sent untouched. The Stac compression algorithm tends to be quite CPU intensive and should not be implemented on routers with an already high CPU utilization. Stac might also be a poor selection on smaller routers that do not have much CPU to begin with.

Predictor

The Predictor compression method is rightly named. This Cisco-proprietary algorithm attempts to predict the incoming character sequences by implementing an indexing system that is based on a compression dictionary. It is essentially a code book that is based on possible data sequences. If a character string is found that matches an entry in the dictionary, the string is replaced with the dictionary entry. That entry is a much shorter sequence of characters. At the remote end, the incoming characters are compared to the data dictionary once again to be decoded. The incoming dictionary strings are replaced with the appropriate (and original) information.

The Predictor compression method is like sign language. Rather than spelling out each individual word (no compression), a single hand motion utilizes an entire word or concept (compression). Because both parties understand the hand motions, successful communication occurs. Conversely, when one of the people involved in the communication does not understand sign language, communication does not occur.

Predictor-like algorithms are used in some voice-compression standards. For example, G.729 and G.729a (CSA-CELP) implementations compress a 64-kbps voice stream into an 8-kbps data stream. These implementations are directly based on the code book/dictionary prediction methodology. One big difference between Predictor (link compression) and voice-compression algorithms (payload compression) is that the voice-compression routines compress only the voice data (payload), not the entire frame.

Remember that Stac is CPU intensive. Predictor, on the other hand, tends to be extremely memory intensive. If the router has not been outfitted with a good amount of RAM, Predictor should not be implemented. However, if RAM is plentiful, Predictor is a compression consideration that can be beneficial.

Payload Compression

Payload compression is exactly what its name implies. Also known as per-VC compression, payload compression compresses only the data portion of the transmission. All L2 headers are left intact.

It cannot be assumed that customer WAN links are all dedicated point-to-point connections (PPP or HDLC). For such circuits, link compression can be used because the provider's WAN switches do not examine any portion of the data being transmitted.

However, payload compression is needed if the WAN switches must examine the data sent from a customer location. WAN technologies such as Frame Relay require that the L2 header information be untouched so that the provider's switches can read it and make forwarding decisions based on it. Any implementation of VCs disallows link compression. In these cases, payload compression is appropriate.

TCP Header Compression

RFC 1144 defines the Van Jacobson algorithm. In doing so, it also defines the algorithm for TCP/IP header compression. The 20-byte IP header and the 20-byte TCP header combination (a total of 40 bytes) is compressed to 2 or 4 (typically 4) bytes to reduce overhead across the network. The L2 header remains intact so that it can be utilized by the appropriate L2 transport.

This type of compression is most beneficial when used with implementations that transmit small packets, such as Voice over IP, Telnet, and so forth. This type of compression can be done on just about any WAN implementation (X.25, Frame Relay, ISDN, and so on).

TCP header compression, as suggested in the name, compresses only the TCP and IP headers. If the data payload is 1000 bytes, then the total package (excluding the L2 frame) would be 1000 + 40 (data + TCP + IP) bytes. TCP header compression would bring this down to 1000 + 2 bytes. However, if the payload is 20 bytes, then 20 + 40 becomes 20 + 2 (a big improvement). Table 15-4 summarizes the benefits. This table does not consider the L2 headers.

Table 15-4 *TCP Header Compression*

Data	TCP Header	IP Header	Total Without Compression	% Overhead	Total With Compression	% Overhead
1000	20	20	1040	3.8%	1002	.2%
20	20	20	60	66.7%	22	9.1%

Note that for packets with larger data portions, the compression provides noticeable improvement (3.8 percent overhead compared to .2 percent). However, for smaller data portions, the improvement is dramatic (66.7 percent overhead compared to 9.1 percent).

As with other forms of compression, TCP header compression must be configured on both ends of the link to ensure a connection. Because the L2 headers are not compressed, TCP header compression can be used on any serial interface, and across WAN clouds that must be able to read the L2 headers during transit.

If any form of compression is used in a Cisco router, the exact same form of compression must be implemented on the other end of the link. Failure to apply the same compression algorithm on each causes all data to fail across the link.

Compression Issues

Compression is not a feature that is simply turned on or off. When selecting the algorithm that is to be utilized for a particular deployment, you should consider the following:

- **Modem compression**—Some modems implement compression. Modems that use MNP5 and V.42bis are not compatible. Although each offers 2 and 4 times compression, they cannot communicate with each other. If you use modem compression, make sure that the modems at both ends of the connection are using a common protocol. If compression is being performed by the modem, do not attempt to configure compression at the router level.

  If modem compression is successfully enabled, then data compression (from the router for example) should not be enabled. Remember that compressing a compressed string might increase the size. Conversely, if compression is performed on the router, then the modem should not attempt any further compression.

- **Data encryption**—Encryption occurs at the network layer where compression is an L2 function. The main purpose of encryption is security. Encryption removes common patterns in data streams. In other words, when Stac tries to find redundant strings, there are none. When Predictor looks into the dictionary for common patterns, there are none. Therefore, the compression is unsuccessful and can actually expand the traffic it was attempting to compress. In such a case, the traffic is sent uncompressed.
- **CPU and memory**—Some algorithms are memory intensive and some are CPU intensive. Thus, before you plan or implement compression, you must know the physical configuration of your router (that is, its RAM and CPU) before ordering additional hardware.

Configuring Compression

To configure compression, there are several commands. Most are technology-specific and fairly intuitive. The **compress** configuration command is used at the interface level (normally a slow serial interface) to select the link-compression algorithm. Remember to configure the same compression type on both ends of the point-to-point link.

```
Router(config-if)# compress [predictor | stac | mppc]
```

For Frame Relay connections, use the **frame-relay payload-compress** interface-level configuration command to enable Stac compression on an interface or a subinterface (payload compression). There are no additional configuration parameters for use with this command, as shown by the following command structure:

```
Router(config-if)# frame-relay payload-compress
```

To enable TCP header compression for a given interface, use the **ip tcp header-compression** command. The command structure is as follows:

```
Router(config-if)# ip tcp header-compression [passive]
```

The **passive** keyword at the end of the command specifies that compression be performed only if packets received on that interface are compressed on arrival.

Foundation Summary

This section is a collection of information that provides a convenient review of many key concepts in this chapter. If you are already comfortable with the topics in this chapter, this summary can help you recall a few details. If you just read this chapter, this review should help solidify some key facts. If you are doing your final preparation before the exam, these tables are a convenient way to review the day before the exam.

Table 15-5 summarizes the various advanced queuing techniques discussed in this chapter.

Table 15-5 *Queuing Summary*

FIFO	Class-Based Weighted Fair	Low-Latency
No configuration	Define flows (**class-map**), policy per flow (**policy-map**), and assign to an interface (service-policy)	Same configuration as CBWFQ, but add a priority queue for delay-sensitive traffic
No priority traffic	Administrator-defined policies	High-priority traffic has its own priority queue—guarantee of service
Traffic dispatched on first-come, first-served basis	Traffic dispatched using WFQ between flows, FIFO within flows	Priority queue is sent first, then remaining queues are WFQ, with FIFO within a queue

Table 15-6 summarizes the various queuing commands discussed in this chapter.

Table 15-6 *Queuing Command Summary*

Command	Function
class-map { **match-all** | **match-any** } *class-map-name*	Creates a data structure to select specific traffic. **match-all** is an AND of all conditions within the **class-map**. **match-any** is an OR of the conditions within the **class-map**.
match	Matches specific traffic within a **class-map** (defined below).
access-group {*number* | *name*}	Matches a numbered or named IP access list.
any	Matches all traffic.
class-map *class-map-name*	Creates a nested **class-map**.

Table 15-6 *Queuing Command Summary (Continued)*

Command	Function
cos *IP-TOS*	Matches the IP ToS bits.
destination-address mac *MAC-address*	Matches a specific destination MAC address.
input-interface *interface*	Matches the interface the traffic arrived on.
ip {**dscp** *value* | **precedence** *value* | **rtp** *start-port port-range*}	Matches various IP header values.
mpls experimental *value*	Matches the MPLS experimental bits.
protocol *value*	Matches any given protocol.
qos-group *value*	Matches traffic from a specific QoS group.
source-address mac *MAC-address*	Matches a specific source MAC address.
policy-map *policy-map-name*	Creates a data structure to reference one or more **class-maps** and is assigned to an interface.
class *class-map-name*	Maps the **class-map** (defined earlier) to the **policy-map**. There are many parameters (below) to reference only specific traffic within the class map.
bandwidth {*Kbps-value* | **percent** *value*}	Defines how traffic is handled during congested situation (CBWFQ).
priority *Kbps-value*	Defines how traffic is handled at all times (LLQ).
queue-limit *#-packets*	Defines how many packets may reside in a particular queue.
service-policy *policy-map-name*	Creates nested policy maps.
shape {average *bps-value* | **max-buffers** *buffer-value* | **peak** *bps-value*}	Defines traffic shaping (buffering) parameters.
police *bps-value*	Defines traffic policing (discarding) parameters.
set {**cos** *value* | **ip** *value* | **mpls experimental** *value* | **qos-group** *value*}	Allows marking of various parameters.
service-policy { **input** | **output** } *policy-map-name*	Assigns a policy map to an interface. Packet queuing is normally done outbound.

Table 15-7 summarizes the various advanced queuing techniques discussed in this chapter.

Table 15-7 *Compression Summary*

Link	Stac	Predictor	Payload	TCP Header
Compresses OSI Layers 2–7	Very CPU intensive	Memory intensive	Compresses OSI Layers 3–7	Compresses OSI Layers 3–4
Two algorithms = Stac and Predictor	Replaces redundant strings with tokens	Replaces text with smaller codes		Compresses ONLY IP and TCP headers
	Does not expand encrypted or compressed data (no redundant strings)	Could possibly expand compressed data		Good for very small data packets
	Open standard	Cisco proprietary	Open standard	
Used across point-to-point circuits only	Can be used on both PPP and HDLC links	Can be used on PPP links only	Can be used on HDLC, PPP, and Frame Relay links	Can be used on any links

Q&A

The questions and scenarios in this book are more difficult than what you will experience on the actual exam. The questions do not attempt to cover more breadth or depth than the exam, but they are designed to make sure that you know the answer. Rather than enabling you to derive the answer from clues hidden inside the question itself, the questions challenge your understanding and recall of the subject.

Hopefully, mastering these questions will help you limit the number of exam questions on which you narrow your choices to two options and then guess.

The answers to these questions can be found in Appendix A.

1. Where on a router is queuing implemented?
2. When should queuing be considered a viable implementation?
3. Should a queuing strategy be implemented on all WAN interfaces?
4. When is WFQ enabled by default?
5. How does CBWFQ differ from WFQ?
6. What is the Cisco IOS command to select and sort traffic into various flows in CBWFQ?
7. What is the Cisco IOS command to assign a policy to one or more flows?
8. What makes LLQ more detailed than CBWFQ?
9. What command is used to create LLQ from a CBWFQ configuration?
10. What is the actual Cisco IOS command to match all traffic from subnet 10.1.1.0 /24 to network 192.168.1.0 /24?
11. What are the actual Cisco IOS commands to match the access list in question 10 into a single group or flow?
12. What are the actual Cisco IOS commands to apply a policy that states that "traffic will get 48 kbps during congestion" to the previous flow?
13. What are the actual Cisco IOS commands used to apply the policy in question 12 to interface serial 0/0?
14. List the types of compression supported by most Cisco routers.
15. When should link compression be implemented?
16. Which type of compression should be utilized on VC-based WAN deployments?

17. What are the two link-compression algorithms, and which one is considered an open standard?
18. When is TCP header compression most effective?
19. When can TCP header compression be implemented?
20. What compression options are possible across a Frame Relay link?