Delay Versus Load

Note that when the load is much less than the capacity of the network, the delay is at a minimum. This minimum delay is composed of propagation delay and processing delay, both of which are negligible. However, when the load reaches the network capacity, the delay increases sharply because we now need to add the waiting time in the queues (for all routers in the path) to the total delay. Note that the delay becomes infinite when the load is greater than the capacity. If this is not obvious, consider the size of the queues when almost no packet reaches the destination, or reaches the destination with infinite delay; the queues become longer and longer. Delay has a negative effect on the load and consequently the congestion. When a packet is delayed, the source, not receiving the acknowledgment, retransmits the packet, which makes the delay, and the congestion, worse.

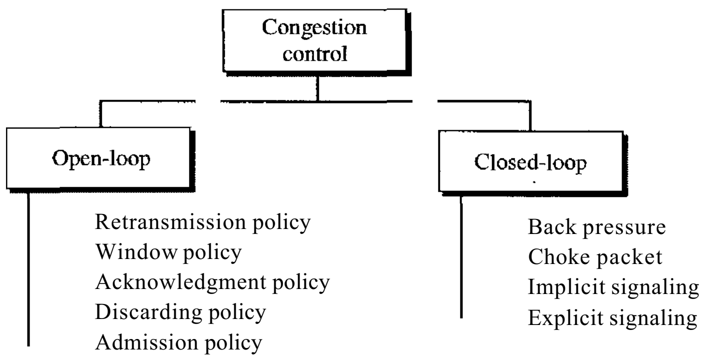
Throughput Versus Load

We defined throughput in Chapter 3 as the number of bits passing through a point in a second. We can extend that definition from bits to packets and from a point to a network. We can define throughput in a network as the number of packets passing through the network in a unit of time. Notice that when the load is below the capacity of the network, the throughput increases proportionally with the *load.* We expect the throughput to remain constant after the load reaches the capacity, but instead the throughput declines sharply. The reason is the discarding of packets by the routers. When the load exceeds the capacity, the queues become full and the routers have to discard some packets. Discarding packets does not reduce the number of packets in the network because the sources retransmit the packets, using time-out mechanisms, when the packets do not reach the destinations.

24.3 CONGESTION CONTROL

Congestion control refers to techniques and mechanisms that can either prevent congestion, before it happens, or remove congestion, after it has happened. In general, we can divide congestion control mechanisms into two broad categories: open-loop congestion control (prevention) and closed-loop congestion control (removal) as shown in Figure 24.5.

Figure 24.5 *Congestion control categories*



Open-Loop Congestion Control

In open-loop congestion control, policies are applied to prevent congestion before it happens. In these mechanisms, congestion control is handled by either the source or the destination. We give a brief list of policies that can prevent congestion.

Retransmission Policy

Retransmission is sometimes unavoidable. If the sender feels that a sent packet is lost or corrupted, the packet needs to be retransmitted. Retransmission in general may increase congestion in the network. However, a good retransmission policy can prevent congestion. The retransmission policy and the retransmission timers must be designed to optimize efficiency and at the same time prevent congestion. For example, the retransmission policy used by TCP (explained later) is designed to prevent or alleviate congestion.

Window Policy

The type of window at the sender may also affect congestion. The Selective Repeat window is better than the Go-Back-N window for congestion control. In the *Go-Back-N* window, when the timer for a packet times out, several packets may be resent, although some may have arrived safe and sound at the receiver. This duplication may make the congestion worse. The Selective Repeat window, on the other hand, tries to send the specific packets that have been lost or corrupted.

Acknowledgment Policy

The acknowledgment policy imposed by the receiver may also affect congestion. If the receiver does not acknowledge every packet it receives, it may slow down the sender and help prevent congestion. Several approaches are used in this case. A receiver may send an acknowledgment only if it has a packet to be sent or a special timer expires. A receiver may decide to acknowledge only *N* packets at a time. We need to know that the acknowledgments are also part of the load in a network. Sending fewer acknowledgments means imposing less load on the network.

Discarding Policy

A good discarding policy by the routers may prevent congestion and at the same time may not harm the integrity of the transmission. For example, in audio transmission, if the policy is to discard less sensitive packets when congestion is likely to happen, the quality of sound is still preserved and congestion is prevented or alleviated.

Admission Policy

An admission policy, which is a quality-of-service mechanism, can also prevent congestion in virtual-circuit networks. Switches in a flow first check the resource requirement of a flow before admitting it to the network. A router can deny establishing a virtual-circuit connection if there is congestion in the network or if there is a possibility of future congestion.

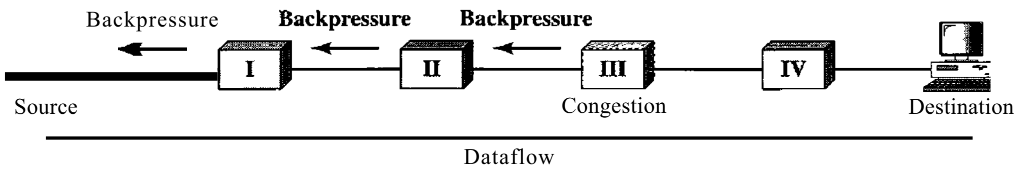
Closed-Loop Congestion Control

Closed-loop congestion control mechanisms try to alleviate congestion after it happens. Several mechanisms have been used by different protocols. We describe a few of them here.

Backpressure

The technique of *backpressure* refers to a congestion control mechanism in which a congested node stops receiving data from the immediate upstream node or nodes. This may cause the upstream node or nodes to become congested, and they, in turn, reject data from their upstream nodes or nodes. And so on. Backpressure is a node-to-node congestion control that starts with a node and propagates, in the opposite direction of data flow, to the source. The backpressure technique can be applied only to virtual circuit networks, in which each node knows the upstream node from which a flow of data is corning. Figure 24.6 shows the idea of backpressure.

Figure 24.6 *Backpressure methodfor alleviating congestion*



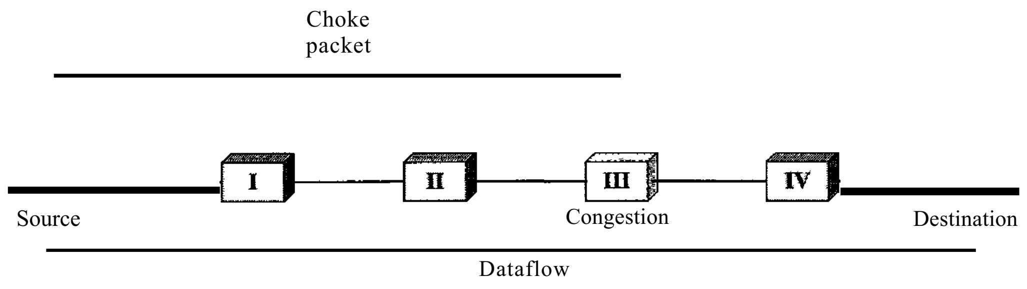
Node III in the figure has more input data than it can handle. It drops some packets in its input buffer and informs node II to slow down. Node II, in turn, may be congested because it is slowing down the output flow of data. If node II is congested, it informs node I to slow down, which in turn may create congestion. If so, node I informs the source of data to slow down. This, in time, alleviates the congestion. Note that the *pressure* on node III is moved backward to the source to remove the congestion.

None of the virtual-circuit networks we studied in this book use backpressure. It was, however, implemented in the first virtual-circuit network, X.25. The technique cannot be implemented in a datagram network because in this type of network, a node (router) does not have the slightest knowledge of the upstream router.

Choke Packet

A choke packet is a packet sent by a node to the source to inform it of congestion. Note the difference between the backpressure and choke packet methods. In backpressure, the warning is from one node to its upstream node, although the warning may eventually reach the source station. In the choke packet method, the warning is from the router, which has encountered congestion, to the source station directly. The intermediate nodes through which the packet has traveled are not warned. We have seen an example of this type of control in ICMP. When a router in the Internet is overwhelmed with IP datagrams, it may discard some of them; but it informs the source host, using a source quench ICMP message. The warning message goes directly to the source station; the intermediate routers, and does not take any action. Figure 24.7 shows the idea of a choke packet.

Figure 24.7 *Choke packet*



Implicit Signaling

In implicit signaling, there is no communication between the congested node or nodes and the source. The source guesses that there is a congestion somewhere in the network from other symptoms. For example, when a source sends several packets and there is no acknowledgment for a while, one assumption is that the network is congested. The delay in receiving an acknowledgment is interpreted as congestion in the network; the source should slow down. We will see this type of signaling when we discuss TCP congestion control later in the chapter.

Explicit Signaling

The node that experiences congestion can explicitly send a signal to the source or destination. The explicit signaling method, however, is different from the choke packet method. In the choke packet method, a separate packet is used for this purpose; in the explicit signaling method, the signal is included in the packets that carry data. Explicit signaling, as we will see in Frame Relay congestion control, can occur in either the forward or the backward direction.

Backward Signaling A bit can be set in a packet moving in the direction opposite to the congestion. This bit can warn the source that there is congestion and that it needs to slow down to avoid the discarding of packets.

Forward Signaling A bit can be set in a packet moving in the direction of the congestion. This bit can warn the destination that there is congestion. The receiver in this case can use policies, such as slowing down the acknowledgments, to alleviate the congestion.

24.4 TWO EXAMPLES

To better understand the concept of congestion control, let us give two examples: one in TCP and the other in Frame Relay.

Congestion Control in TCP

We discussed TCP in Chapter 23. We now show how TCP uses congestion control to avoid congestion or alleviate congestion in the network.

Congestion Window

In Chapter 23, we talked about flow control and tried to discuss solutions when the receiver is overwhelmed with data. We said that the sender window size is determined by the available buffer space in the receiver *(rwnd).* In other words, we assumed that it is only the receiver that can dictate to the sender the size of the sender's window. We totally ignored another entity here-the network. If the network cannot deliver the data as fast as they are created by the sender, it must tell the sender to slow down. In other words, in addition to the receiver, the network is a second entity that determines the size of the sender's window.

Today, the sender's window size is determined not only by the receiver but also by congestion in the network.

The sender has two pieces of information: the receiver-advertised window size and the congestion window size. The actual size of the window is the minimum of these two.

Actual window **size** = minimum (rwnd, cwnd)

We show shortly how the size of the congestion window *(cwnd)* is determined.

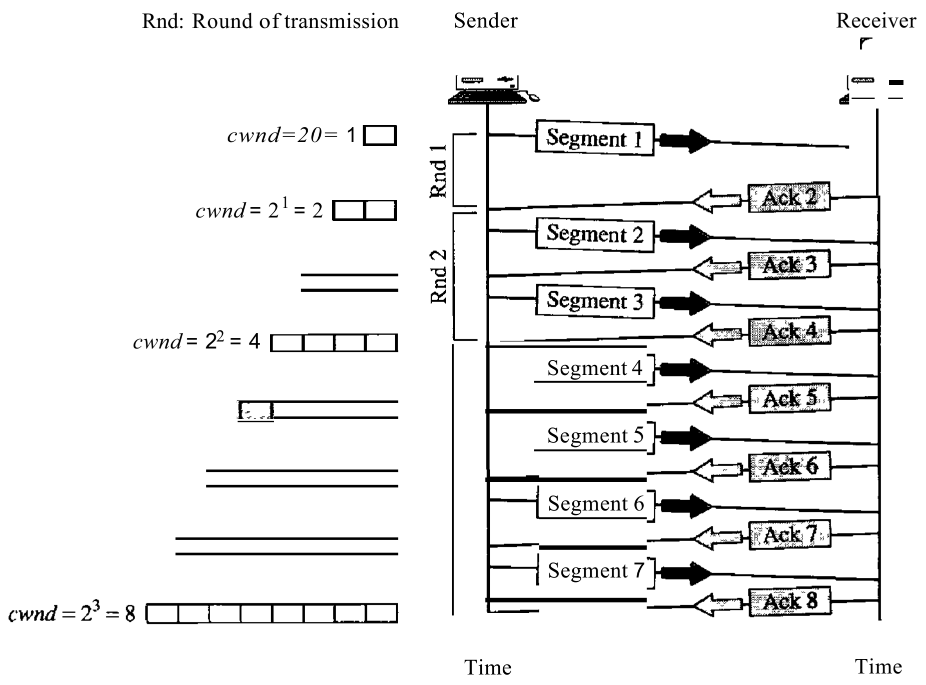
Congestion Policy

TCP's general policy for handling congestion is based on three phases: slow start, congestion avoidance, and congestion detection. In the slow-start phase, the sender starts with a very slow rate of transmission, but increases the rate rapidly to reach a threshold. When the threshold is reached, the data rate is reduced to avoid congestion. Finally if congestion is detected, the sender goes back to the slow-start or congestion avoidance phase based on how the congestion is detected.

Slow Start: Exponential Increase One of the algorithms used in TCP congestion control is called slow start. This algorithm is based on the idea that the size of the congestion window *(cwnd)* starts with one maximum segment size (MSS). The MSS is determined during connection establishment by using an option of the same name. The size of the window increases one MSS each time an acknowledgment is received. As the name implies, the window starts slowly, but grows exponentially. To show the idea, let us look at Figure 24.8. Note that we have used three simplifications to make the discussion more understandable. We have used segment numbers instead of byte numbers (as though each segment contains only 1 byte). We have assumed that *rwnd* is much higher than *cwnd,* so that the sender window size always equals *cwnd.* We have assumed that each segment is acknowledged individually.

The sender starts with *cwnd* = 1 MSS. This means that the sender can send only one segment. After receipt of the acknowledgment for segment 1, the size of the congestion window is increased by 1, which means that *cwnd* is now 2. Now two more segments can be sent. When each acknowledgment is received, the size of the window is increased by 1 MSS. When all seven segments are acknowledged, *cwnd* = 8.

Figure 24.8 *Slow start, exponential increase*



If we look at the size of *cwnd* in terms of rounds (acknowledgment of the whole window of segments), we find that the rate is exponential as shown below:

Start	➡	*cwnd=1*
After round 1	➡	$cwnd=2^1=2$
After round 2	➡	$cwnd=2^2=4$
After round 3	➡	$cwnd=2^3=8$

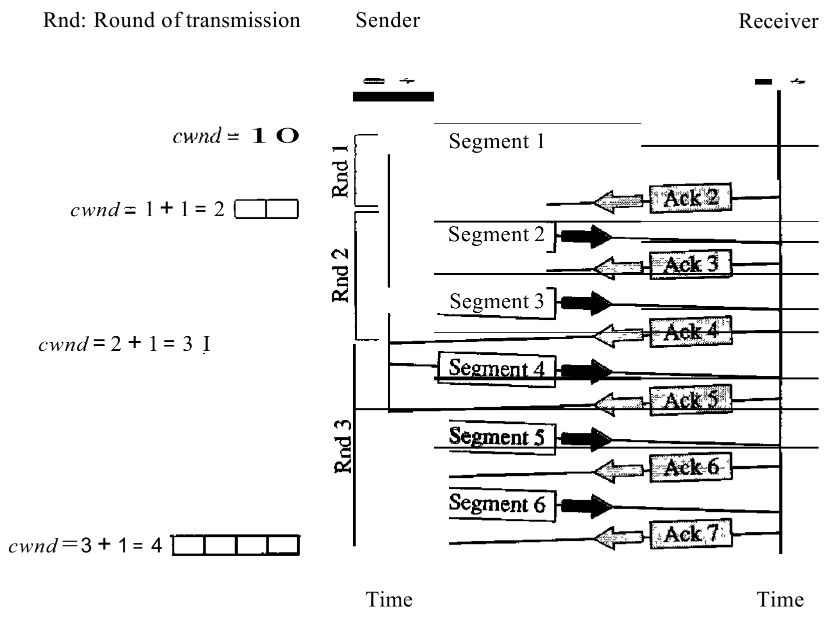
We need to mention that if there is delayed ACKs, the increase in the size of the window is less than power of 2.

Slow start cannot continue indefinitely. There must be a threshold to stop this phase. The sender keeps track of a variable named *ssthresh* (slow-start threshold). When the size of window in bytes reaches this threshold, slow start stops and the next phase starts. In most implementations the value of *ssthresh* is 65,535 bytes.

In the slow-start algorithm, the size of the congestion window increases exponentially until it reaches a threshold.

Congestion Avoidance: Additive Increase If we start with the slow-start algorithm, the size of the congestion window increases exponentially. To avoid congestion before it happens, one must slow down this exponential growth. TCP defines another algorithm called congestion avoidance, which undergoes an additive increase instead of an exponential one. When the size of the congestion window reaches the slow-start threshold, the slow-start phase stops and the additive phase begins. In this algorithm, each time the whole window of segments is acknowledged (one round), the size of the congestion window is increased by 1. To show the idea, we apply this algorithm to the same scenario as slow start, although we will see that the congestion avoidance algorithm usually starts when the size of the window is much greater than 1. Figure 24.9 shows the idea.

Figure 24.9 *Congestion avoidance, additive increase*



In this case, after the sender has received acknowledgments for a complete window size of segments, the size of the window is increased by one segment.

If we look at the size of *cwnd* in terms of rounds, we find that the rate is additive as shown below:

Start	➡	*cwnd*=1
After round 1	➡	*cwnd*= 1+1=2
After round 2	➡	*cwnd*=2+1=3
After round 3	➡	*cwnd*=3+1=4

In the congestion avoidance algorithm, the size of the congestion window increases additively until congestion is detected.

Congestion Detection: Multiplicative Decrease If congestion occurs, the congestion window size must be decreased. The only way the sender can guess that congestion has occurred is by the need to retransmit a segment. However, retransmission can occur in one of two cases: when a timer times out or when three ACKs are received. In both cases, the size of the threshold is dropped to one-half, a multiplicative decrease. Most TCP implementations have two reactions:

1. If a time-out occurs, there is a stronger possibility of congestion; a segment has probably been dropped in the network, and there is no news about the sent segments.

In this case TCP reacts strongly:

a. It sets the value of the threshold to one-half of the current window size.

b. It sets *cwnd* to the size of one segment.

c. It starts the slow-start phase again.

2. If three ACKs are received, there is a weaker possibility of congestion; a segment may have been dropped, but some segments after that may have arrived safely since three ACKs are received. This is called fast transmission and fast recovery. In this case, TCP has a weaker reaction:

a. It sets the value of the threshold to one-half of the current window size.

b. It sets *cwnd* to the value of the threshold (some implementations add three segment sizes to the threshold).

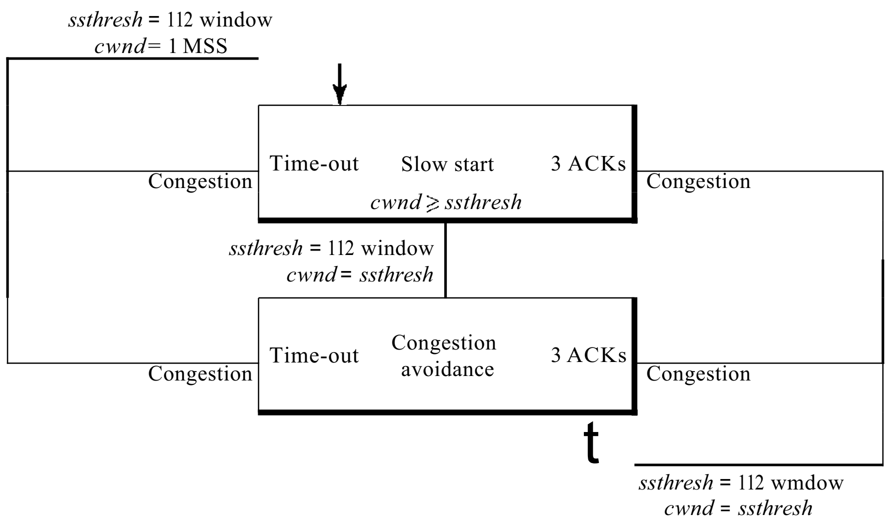
c. It starts the congestion avoidance phase.

An implementations reacts to congestion detection in one of the following ways:

- If detection is by time-out, a new *slow-start* phase starts.
- If detection is by three ACKs, a new *congestion avoidance* phase starts.

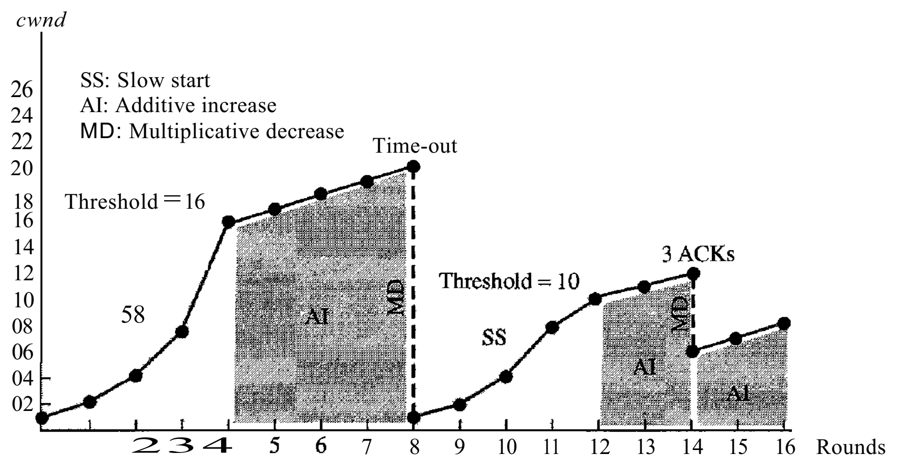
Summary In Figure 24.10, we summarize the congestion policy of TCP and the relationships between the three phases.

Figure 24.10 *TCP congestion policy summary*



We give an example in Figure 24.11. We assume that the maximum window size is 32 segments. The threshold is set to 16 segments (one-half of the maximum window size). In the *slow-start* phase the window size starts from 1 and grows exponentially until it reaches the threshold. After it reaches the threshold, the *congestion avoidance (additive increase)* procedure allows the window size to increase linearly until a time-out occurs or the maximum window size is reached. In Figure 24.11, the time-out occurs when the window size is 20. At this moment, the *multiplicative decrease* procedure takes

Figure 24.11 *Congestion example*



over and reduces the threshold to one-half of the previous window size. The previous window size was 20 when the time-out happened so the new threshold is now 10.

TCP moves to slow start again and starts with a window size of 1, and TCP moves to additive increase when the new threshold is reached. When the window size is 12, a three-ACKs event happens. The multiplicative decrease procedure takes over again. The threshold is set to 6 and TCP goes to the additive increase phase this time. It remains in this phase until another time-out or another three ACKs happen.

Congestion Control in Frame Relay

Congestion in a Frame Relay network decreases throughput and increases delay. A high throughput and low delay are the main goals of the Frame Relay protocol. Frame Relay does not have flow control. In addition, Frame Relay allows the user to transmit bursty data. This means that a Frame Relay network has the potential to be really congested with traffic, thus requiring congestion control.

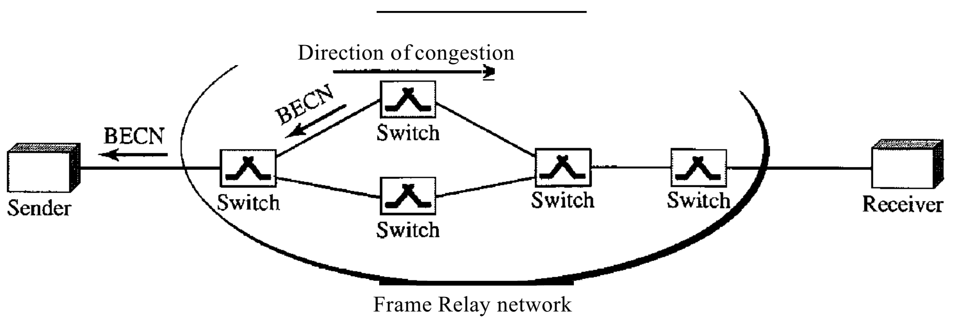
Congestion Avoidance

For congestion avoidance, the Frame Relay protocol uses 2 bits in the frame to explicitly warn the source and the destination of the presence of congestion.

BECN The backward explicit congestion notification (BECN) bit warns the sender of congestion in the network. One might ask how this is accomplished since the frames are traveling away from the sender. In fact, there are two methods: The switch can use response frames from the receiver (full-duplex mode), or else the switch can use a predefined connection (DLCI = 1023) to send special frames for this specific purpose. The sender can respond to this warning by simply reducing the data rate. Figure 24.12 shows the use ofBECN.

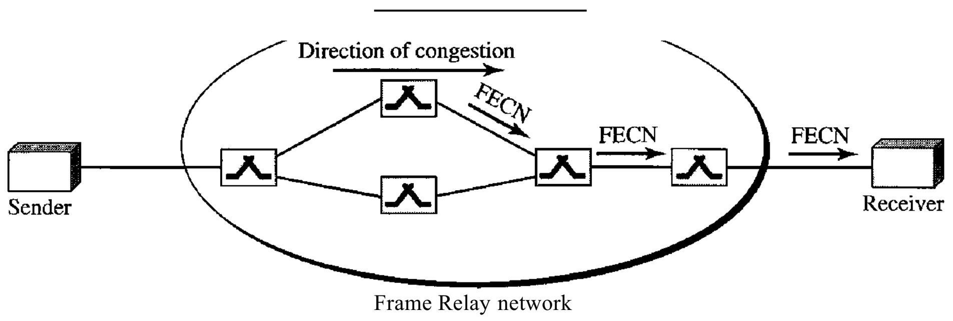
FECN The forward explicit congestion notification (FECN) bit is used to warn the receiver of congestion in the network. It might appear that the receiver cannot do anything to relieve the congestion. However, the Frame Relay protocol assumes that the sender and receiver are communicating with each other and are using some type of flow control at a higher level. For example, if there is an acknowledgment mechanism at this

Figure 24.12 *BECN*



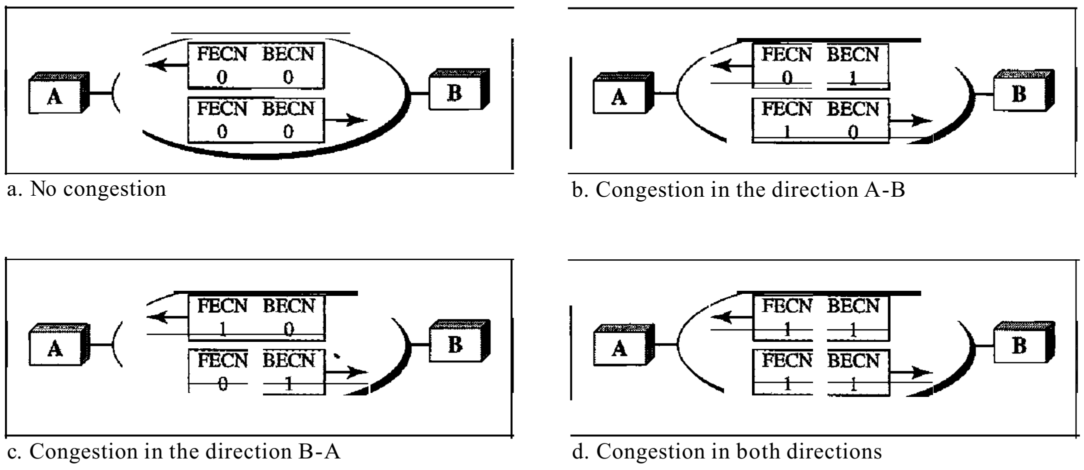
higher level, the receiver can delay the acknowledgment, thus forcing the sender to slow down. Figure 24.13 shows the use ofFECN.

Figure 24.13 *FECN*



When two endpoints are communicating using a Frame Relay network, four situations may occur with regard to congestion. Figure 24.14 shows these four situations and the values of FECN and BECN.

Figure 24.14 *Four cases ofcongestion*



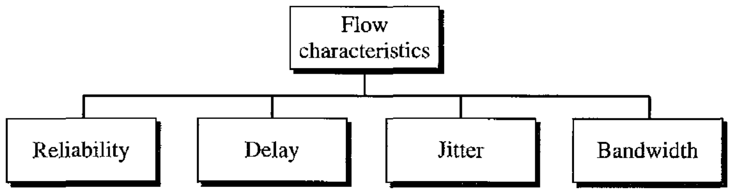
24.5 QUALITY OF SERVICE

Quality of service (QoS) is an internetworking issue that has been discussed more than defined. We can informally define quality of service as something a flow seeks to attain.

Flow Characteristics

Traditionally, four types of characteristics are attributed to a flow: reliability, delay, jitter, and bandwidth, as shown in Figure 24.15.

Figure 24.15 *Flow characteristics*



Reliability

Reliability is a characteristic that a flow needs. Lack of reliability means losing a packet or acknowledgment, which entails retransmission. However, the sensitivity of application programs to reliability is not the same. For example, it is more important that electronic mail, file transfer, and Internet access have reliable transmissions than telephony or audio conferencing.

Delay

Source-to-destination delay is another flow characteristic. Again applications can tolerate delay in different degrees. In this case, telephony, audio conferencing, video conferencing, and remote log-in need minimum delay, while delay in file transfer or e-mail is less important.

Jitter

Jitter is the variation in delay for packets belonging to the same flow. For example, if four packets depart at times 0, 1, 2, 3 and arrive at 20, 21, 22, 23, all have the same delay, 20 units of time. On the other hand, if the above four packets arrive at 21, 23, 21, and 28, they will have different delays: 21,22, 19, and 24.

For applications such as audio and video, the first case is completely acceptable; the second case is not. For these applications, it does not matter if the packets arrive with a short or long delay as long as the delay is the same for all packets. For this application, the second case is not acceptable.

Jitter is defined as the variation in the packet delay. High jitter means the difference between delays is large; low jitter means the variation is small.

In Chapter 29, we show how multimedia communication deals with jitter. If the jitter is high, some action is needed in order to use the received data.

Bandwidth

Different applications need different bandwidths. In video conferencing we need to send millions of bits per second to refresh a color screen while the total number of bits in an e-mail may not reach even a million.

Flow Classes

Based on the flow characteristics, we can classify flows into groups, with each group having similar levels of characteristics. This categorization is not formal or universal; some protocols such as ATM have defined classes, as we will see later.

24.6 lECHNIQUES TO IMPROVE QoS

In Section 24.5 we tried to define QoS in terms of its characteristics. In this section, we discuss some techniques that can be used to improve the quality of service. We briefly discuss four common methods: scheduling, traffic shaping, admission control, and resource reservation.

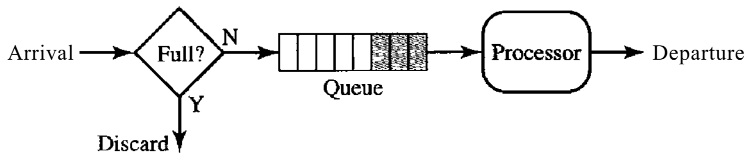
Scheduling

Packets from different flows arrive at a switch or router for processing. A good scheduling technique treats the different flows in a fair and appropriate manner. Several scheduling techniques are designed to improve the quality of service. We discuss three of them here: FIFO queuing, priority queuing, and weighted fair queuing.

FIFO Queuing

In first-in, first-out (FIFO) queuing, packets wait in a buffer (queue) until the node (router or switch) is ready to process them. If the average arrival rate is higher than the average processing rate, the queue will fill up and new packets will be discarded. A FIFO queue is familiar to those who have had to wait for a bus at a bus stop. Figure 24.16 shows a conceptual view of a FIFO queue.

Figure 24.16 *FIFO queue*



Priority Queuing

In priority queuing, packets are first assigned to a priority class. Each priority class has its own queue. The packets in the highest-priority queue are processed first. Packets in the lowest-priority queue are processed last. Note that the system does not stop serving