CHAPTER 7

Packet-Switching Networks

Traditional telephone networks operate on the basis of circuit switching. A call setup process reserves resources (time slots) along a path so that the stream of voice samples can be transmitted with very low delay across the network. The resources allocated to a call cannot be used by other users for the duration of the call. This approach is inefficient when the amount of informtion transferred is small or if information is produced in bursts, as is the case in many computer applications. In this chapter we examine networks that transfer blocks of information called **packets**. **Packet-switching networks** are better matched to computer applications and can also be designed to support real-time applications such as telephony.

We can view packet networks from two perspectives. One perspective involves an *external view* of the network and is concerned with the services that the network provides to the transport layer that operates above it at the end systems. Here we are concerned with whether the network service requires the setting up of a connection and whether the transfer of user data is provided with quality-of-service guarantees. Ideally the definition of the network services is independent of the underlying network and transmission technologies. This approach allows the transport layer and the applications that operate above it to be designed so that they can function over any network that provides the given services.

A second perspective on packet networks is concerned with the *internal* operation of a network. Here we look at the physical topology of a network, the interconnection of links, switches, and routers. We are concerned with the approach that is used to direct information across the network: datagrams, or virtual circuits. We are also concerned with addressing and routing procedures, as well as with dealing with congestion inside the network. We must also manage traffic flows so that the network can deliver information with the quality of service it has committed to.

It is useful to compare these two perspectives in the case of broadcast networks and LANs from the previous chapter and the switched packet networks considered here. The first perspective, involving the services provided to the layer above, does not differ in a fundamental way between broadcast and switched packet networks. The second perspective, however, is substantially different. In the case of LANs, the network is small, addressing is simple, and the frame is transferred in one hop so no routing is required. In the case of packet-switching networks, addressing must accommodate extremely large-scale networks and must work in concert with appropriate routing algorithms. These two challenges, addressing and routing, are the essence of the network layer.

In this chapter we deal with the general issues regarding packet-switching networks. Later chapters deal with specific architectures, namely, Internet Protocol (IP) packet networks and asynchronous transfer mode (ATM) packet networks. The chapter is organized as follows:

1. *Network services and internal network operation*. We elaborate on the two perspectives on networks, and we discuss the functions of the network layer, including internetworking.
2. *Physical view of networks*. We examine typical configurations of packet-switching networks. This section defines the role of multiplexers, LANs, switches, and routers in network and internetwork operation.
3. *Datagrams and virtual circuits*. We introduce the two basic approaches to operating a packet network, and we use IP and ATM as examples of these approaches.
4. *Routing*. We introduce the basic approaches for selecting routes across the network.
5. *Shortest path algorithms*. We continue our discussion of routing, focusing on two shortest-path routing algorithms: the Bellman-Ford algorithm and Dijskstra's algorithm.
6. *ATM networks*. We introduce ATM networks as an example of an advanced virtual-circuit packet-switching network that can support many services.
7. *Traffic management*. We introduce traffic shaping, scheduling and call admission control as methods for providing Quality-of-Service.
8. *Congestion control*. We introduce techniques to deal with congestion due to surges in traffic or equipment failures.

The material on ATM, traffic management, and congestion control is relatively advanced. The corresponding sections (7.6, 7.7, and 7.8 respectively) can be skipped and the reader may proceed to Chapter 8, depending on their background or interest.

7.1 NETWORK SERVICES AND INTERNAL NETWORK OPERATION

The essential function of a network is to transfer information among the users that are attached to the network or internetwork. In Figure 7.1 we show that this transfer may involve a single block of information or a sequence of blocks that are temporally related. In the case of a single block of information, we are interested in having the block delivered correctly to the destination, and we may also be interested in the delay experienced in traversing the network. In the case of a sequence of blocks, we may be interested not only in receiving the blocks correctly and in the right sequence but also in delivering a relatively unimpaired temporal relation.

Figure 7.2 shows a transport protocol that operates end to end across a network. The transport layer peer processes at the end systems accept messages from their higher layer and transfer these messages by exchanging segments end to end across the network. The figure shows the interface at which the network service is visible to the transport layer. The network service is all that matters to the transport layer, and the manner in which the network operates to provide the service is irrelevant.

The network service can be connection-oriented or connectionless. A connectionless service is very simple, with only two basic interactions between the transport layer and the network layer: a request to the network that it send a packet and an indication from the network that a packet has arrived. The user can request transmission of a packet at any time, and *does not need to inform the network layer* that the user intends to transmit information ahead of time. A connectionless service puts total responsibility for error control, sequencing, and flow control on the end-system transport layer.

The network service can be connection-oriented. In this case the transport layer cannot request transmission of information until a connection has been set up. The essential points here are that *the network layer must be informed* about the new flow that is about to be applied to the network and that the network layer maintains state information about the flows it is handling. During call setup, parameters related to usage and quality of service may be negotiated and network resources may be allocated to ensure that the user flow can be handled as required. A connection-release procedure may also be required to

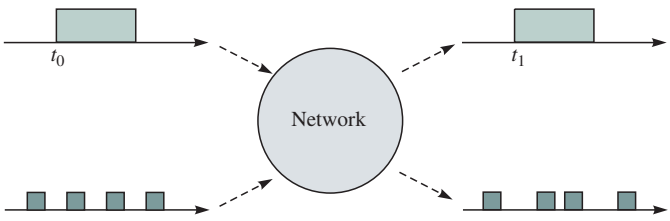


FIGURE 7.1 A network transfers information among users

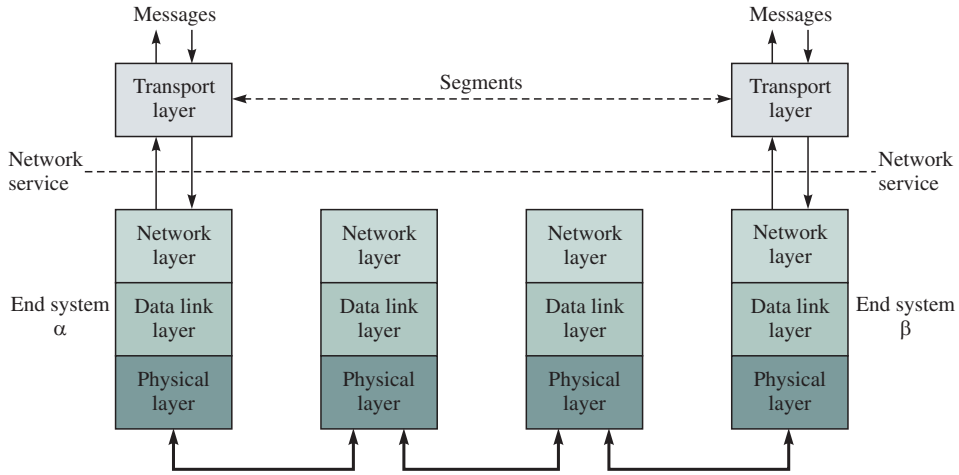


FIGURE 7.2 Peer-to-peer protocols operating end to end across a network—protocol stack view

terminate the connection. It is clear that providing connection-oriented service entails greater complexity than connectionless service in the network layer.

It is also possible for a network layer to provide a choice of services to the user of the network. For example, the network layer could offer: (1) best-effort connectionless service; (2) low-delay connectionless service; (3) connection-oriented reliable stream service; and (4) connection-oriented transfer of packets with delay and bandwidth guarantees. It is easy to come up with examples of applications that can make use of each of these services. However, it does not follow that all the services should be offered by the network layer. Two interrelated reasons can be given for keeping the set of network services to a minimum: the end-to-end argument and the need for network scalability.

When applied to the issue of choice of network services, the end-to-end argument suggests that functions should be placed as close to the application as possible, since it is the application that is in the best position to determine whether a function is being carried out completely and correctly. This argument suggests that as much functionality as possible should be located in the transport layer or higher and that the network services should provide the minimum functionality required to meet application performance.

Up to this point we have considered only the services offered by the network layer. Let us now consider the internal operation of the network. Figure 7.3 shows the relation between the service offered by the network and the internal operation. We say that the internal operation of a network is *connectionless* if packets are transferred within the network as datagrams. Thus in the figure each packet is routed independently. Consequently packets may follow different paths from α to β and so may arrive out of order. We say that the internal operation of a network is *connection-oriented* if packets follow virtual circuits that have been

THE END-TO-END ARGUMENT FOR SYSTEM DESIGN

The *end-to-end argument* in system design articulated in [Saltzer 1984] states that an end-to-end function is best implemented at a higher level than at a lower level. The reason is that the correct end-to-end implementation requires *all* intermediate low-level components to operate correctly. This feature is difficult and sometimes impossible to ensure and is frequently too costly. The higher-level components at the ends are in a better position to determine that a function has been carried out correctly and in better position to take corrective action if they have not. Low-level actions to support the end-to-end function are justified only as performance enhancements.

We already encountered the end-to-end argument in the comparison of end-to-end error control and hop-by-hop error control in Chapter 5. The argument here is that the end system will have to implement error control on an end-to-end basis regardless of lower-level error-control mechanisms that may be in place because the individual low-level mechanisms cannot cover all sources of errors, for example, errors introduced within a node. Consequently, lower-level mechanisms are not essential and should be introduced only to enhance performance. Thus the transmission of a long file over a sequence of nearly error-free links does not require per link error control. On the other hand, the transmission of such files over a sequence of error-prone links does argue for per link error control.

established from a source to a destination. Thus to provide communications between α and β, routing to set up a virtual circuit is done once, and thereafter packets are simply forwarded along the established path. If resources are reserved during connection setup, then bandwidth, delay, and loss guarantees can be provided.

The fact that a network offers connection-oriented service, connectionless service, or both does not dictate how the network must operate internally. In discussing TCP and IP, we have already seen that a connectionless packet network (e.g., IP) can support connectionless service (UDP) as well as connection-oriented service (TCP). We will also see that a connection-oriented network (e.g., ATM) can provide connectionless service as well as connection-oriented service. We discuss virtual-circuit and datagram network operation in more detail in a later section. However, it is worthwhile to compare the two at this point at a high level.

The approach suggested by the end-to-end argument keeps the network service (and the network layer that provides the service) as simple as possible while adding complexity at the edge only as required. This strategy fits very well with the need to grow networks to very large scale. We have seen that the value of a network grows with the community of users that can be reached and with the range of applications that can be supported. Keeping the core of the network simple and adding the necessary complexity at the edge enhances the scalability of the network to larger size and scope.

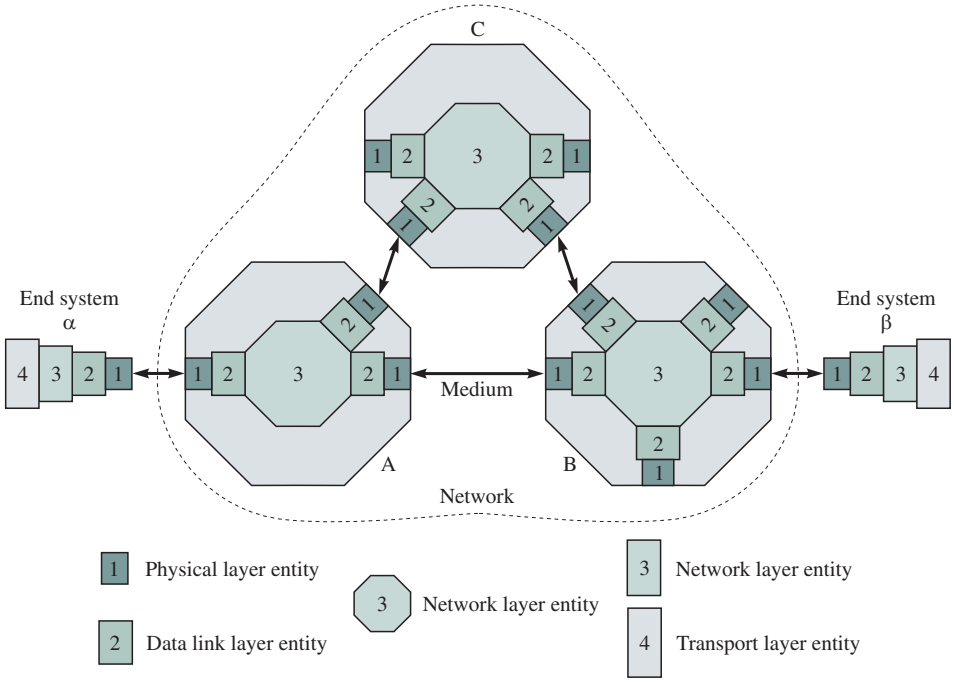


FIGURE 7.3 Layer 3 entities work together to provide network service to layer 4 entities

This reasoning suggests a preference for a connectionless network, which has much lower complexity than a connection-oriented network. The reasoning does allow the possibility for some degree of "connection orientation" as a means to ensure that applications can receive the proper level of performance. Indeed current research and standardization efforts (discussed in Chapter 10) can be viewed as an attempt in this direction to determine an appropriate set of network services and an appropriate mode of internal network operation.

We have concentrated on high-level arguments up to this point. What do these arguments imply about the functions that should be in the network layer? Clearly, functions that need to be carried out at every node in the network must be in the network layer. Thus functions that route and forward packets need to be done in the network layer. Priority and scheduling functions that direct how packets are forwarded so that quality of service is provided also need to be in the network layer. Functions that belong in the edge should, if possible, be implemented in the transport layer or higher. A third category of functions can be implemented either at the edge or inside the network. For example, while congestion takes place inside the network, the remedy involves reducing input flows at the edge of the network. We will see that congestion control has been implemented in the transport layer and in the network layer.

Another set of functions is concerned with making the network service independent of the underlying transmission systems. For example, different

transmissions sytems (e.g., optical versus wireless) may have different limits on the frame size they can handle. The network layer may therefore be called upon to carry out segmentation inside the network and reasssembly at the edge. Alternatively, the network could send error messages to the sending edge, requesting that the packet size be reduced. A more challenging set of functions arises when the "network" itself may actually be an internetwork. In this case the network layer must also be concerned not only about differences in the size of the units that the component networks can transfer but also about differences in addressing and in the services that the component networks provide.

In the remainder of the chapter we deal with the general aspects of internal network operation. In Chapters 8 and 9 we discuss the specific details of IP and ATM networks.

7.2 PACKET NETWORK TOPOLOGY

This section considers existing packet-switching networks. We present an end-to-end view of existing networks from a personal computer, workstation, or server through LANs and the Internet and back.

First let us consider the way in which users access packet networks. Figure 7.4 shows an *access multiplexer* where the packets from a number of users share a transmission line. This system arises for example, in X.25, frame relay, and ATM networks, where a single transmission line is shared in the access to a wide area packet-switching network. The multiplexer combines the typically bursty flows of the individual computers into aggregated flows that make efficient use of the transmission line. Note that different applications within a single computer can generate multiple simultaneous flows to different destinations. From a logical point of view, the link can be viewed as carrying either a single aggregated flow or a number of separate packet flows. The network access node forwards packets into a backbone packet network.

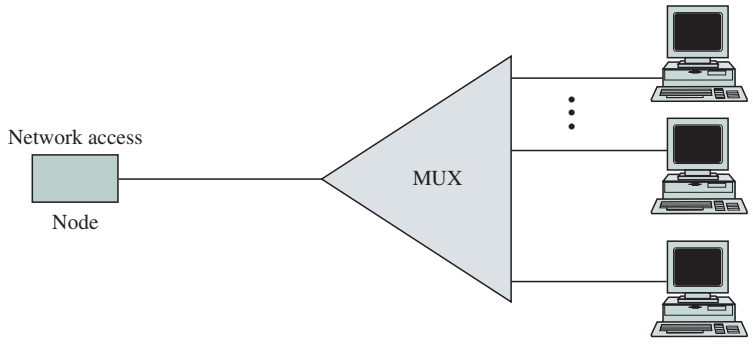


FIGURE 7.4 Access multiplexer

Local area networks (LANs) provide the access to packet-switching networks in many environments. As shown in Figure 7.5a, computers are connected to a shared transmission medium. Transmissions are broadcast to all computers in the network. Each computer is identified by a unique physical address, and so each station listens for its address to receive transmissions. Broadcast and multicast transmissions are easily provided in this environment.

LANs allow the sharing of resources such as printers, databases, and software among a small community of users. LANs can be extended through the use of *bridges* or *LAN switches*, as shown in Figure 7.5b. Here the LAN switch forwards inter-LAN traffic based on the physical address of the frames. Traffic local to each LAN stays local, and broadcast transmissions are forwarded to the other attached LANs. Switches can interconnect more than two LANs.

Multiple LANs in an organization, in turn, are interconnected into *campus networks* with a structure such as that shown in Figure 7.6. LANs for a large group of users such as a department are interconnected in an extended LAN through the use of LAN switches, identified by lowercase *s* in the figure. Resources such as servers and databases that are primarily of use to this department are kept within the subnetwork. This approach reduces delays in accessing the resources and contains the level of traffic that leaves the subnetwork. Each subnetwork has access to the rest of the organization through a router *R* that accesses the campus backbone network. A subnetwork also uses the campus backbone to reach the "outside world" such as the Internet or other sites belonging to the organization through a gateway router. Depending on the type of organization, the gateway may implement firewall functions to control the traffic that is allowed into and out of the campus network.

Servers containing critical resources that are required by the entire organization are usually located in a data center where they can be easily maintained and

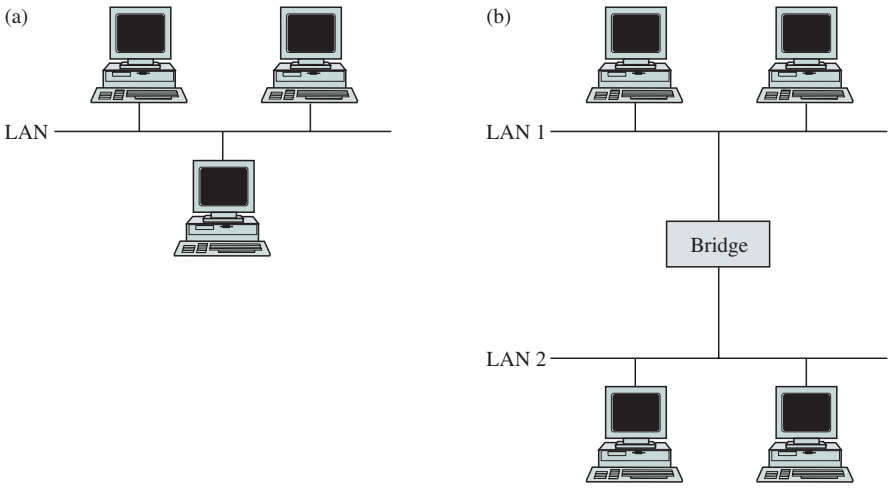


FIGURE 7.5 Local area networks

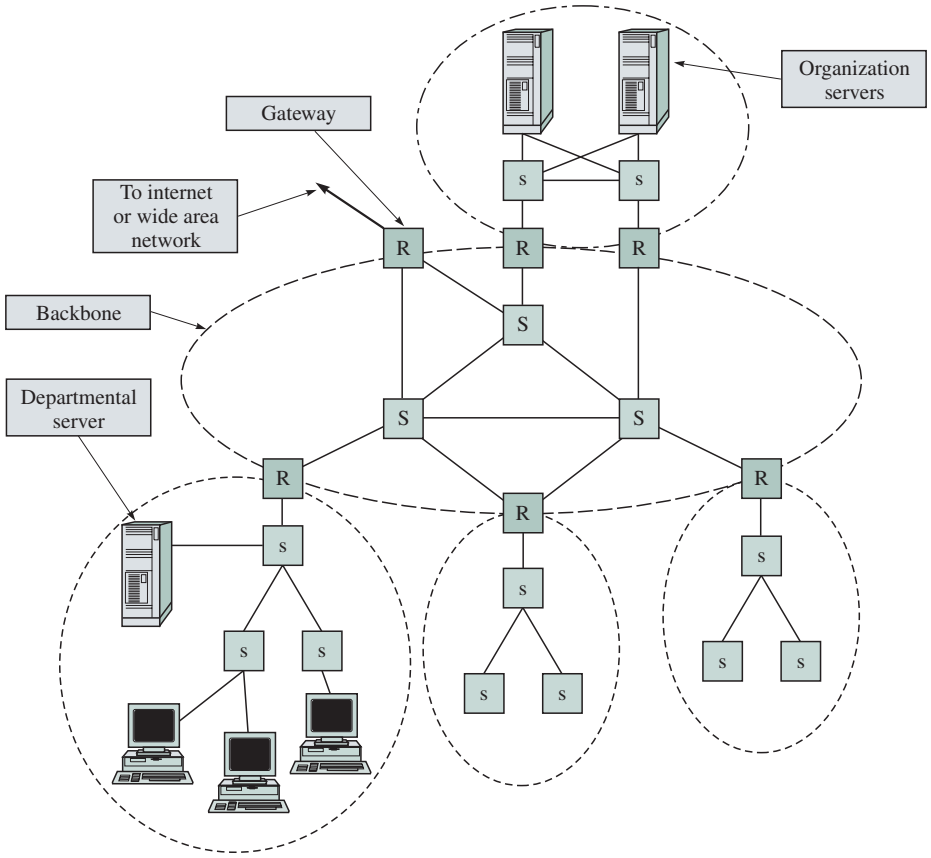


FIGURE 7.6 Campus network

where security can be enforced. As shown in Figure 7.6, the critical servers may be provided with redundant paths to the campus backbone network. These servers are usually placed near the backbone network to minimize the number of hops required to access them from the rest of the organization.

The traffic within an extended LAN is delivered based on the *physical* LAN addresses. However, applications in host computers operate on the basis of *logical* IP addresses. Therefore, the physical address corresponding to an IP address needs to be determined every time an IP packet is to be transmitted over a LAN. This *address resolution* problem can be solved by using IP address to physical address translation tables. In the next chapter we discuss the *Address Resolution Protocol* that IP uses to solve this problem.

The routers in the campus network are interconnected to form the campus backbone network, depicted by the mesh of switches, designated *S*, in Figure 7.6. Typically, for large organizations such as universities these routers are interconnected by using very high speed LANs, for example, Gigabit Ethernet or an

ATM network. The routers use the Internet Protocol (IP), which enables them to operate over various data link and network technologies. The routers exchange information about the state of their links to dynamically calculate routing tables that direct packets across the campus network. This approach allows the network to adapt to changes in traffic pattern as well as changes in topology due to faults in equipment.

The routers in the campus network form a *domain* or *autonomous system*. The term *domain* indicates that the routers run the same routing protocol. The term *autonomous* system is used for one or more domains under a single administration. All routing decisions inside the autonomous system are independent of any other network.

Organizations with multiple sites may have their various campus networks interconnected through routers interconnected by leased digital transmission lines or frame relay connections. In this case access to the *wide area network* may use an access multiplexer such as the one shown in Figure 7.4. In addition the campus network may be connected to an *Internet service provider* through one or more border routers as shown in Figure 7.7. To communicate with other networks, the autonomous system must provide information about its network routes in the border routers. The border router communicates on an interdomain level, whereas other routers in a campus network operate at the intradomain level.

A national ISP provides points of presence (POPs) in various cities where customers can connect to their network. The ISP has its own national network for interconnecting its POPs. This network could be based on ATM; it might use IP over SONET; or it might use some other network technology. The ISPs in turn exchange traffic as *network access points (NAPs)*, as shown in Figure 7.8a. A NAP is a high-speed LAN or switch at which the routers from different ISPs

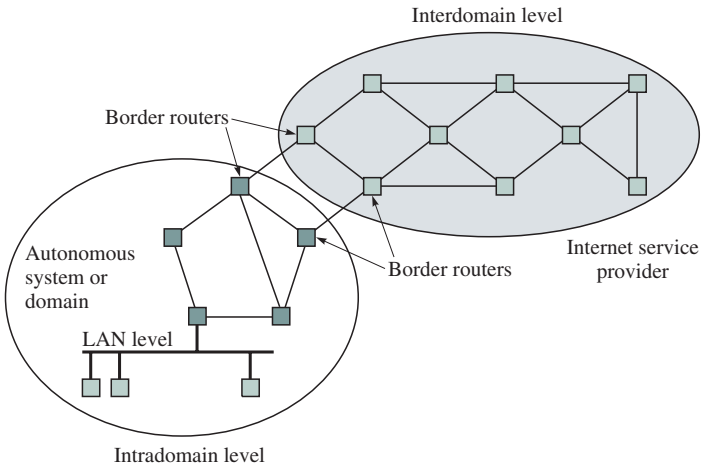


FIGURE 7.7 Intradomain and interdomain levels

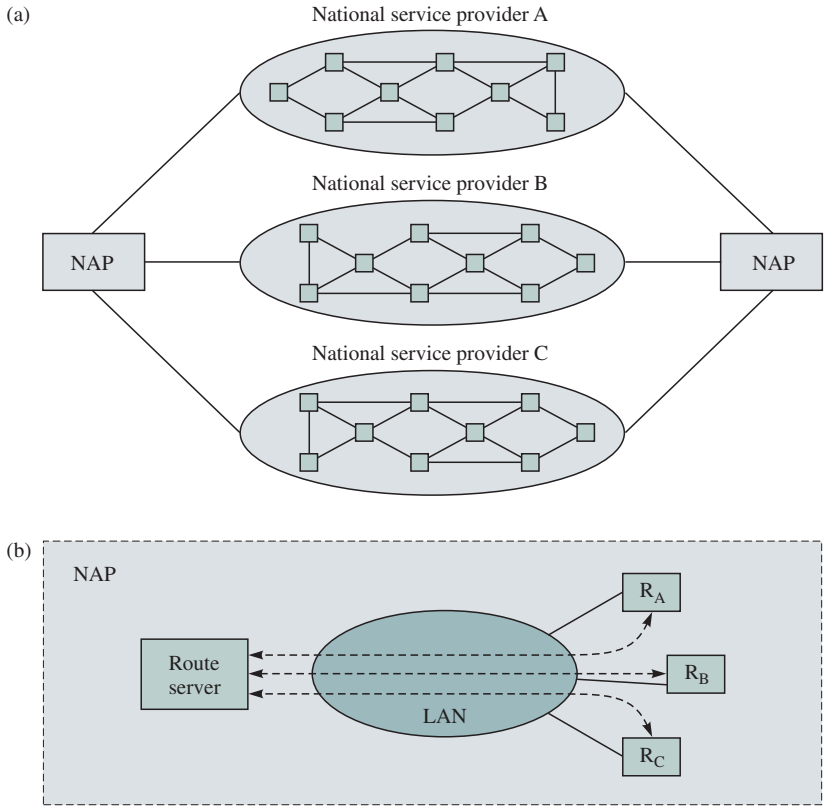


FIGURE 7.8 National ISPs exchange traffic at NAPs; routing information is exchanged through route servers

can exchange traffic, and as such NAPs are crucial to the interconnectivity provided by the Internet. (As discussed in Chapter 1, four NAPs were originally set up by the National Science Foundation). The ISPs interconnected to a NAP need to exchange routing information. If there are n such ISPs, then $n(n-1)$ pairwise route exchanges are required. This problem is solved by introducing a route server as shown in Figure 7.8b. Each ISP sends routing information to the route server, which knows the policies of every ISP. The route server in turn delivers the processed routing information to the ISPs.

Note that a national service provider also has the capability of interconnecting a customer's various sites by using its own IP network, so the customer's sites appear as a single private network. This configuration is an example of a virtual private network (VPN).

Small office and home (SOHO) users obtain packet access through ISPs. The access is typically through modem dial-up, but it could be through ADSL, ISDN, or cable modem. When a customer connects to an ISP, the customer is

assigned an IP address for the duration of the connection.[1] Addresses are shared in this way because the ISP has only a limited number of addresses. If the ISP is only a local provider, then it must connect to a regional or national provider and eventually to a NAP.

Thus we see that a multilevel hierarchical network topology arises for the Internet which is much more decentralized than traditional telephone networks. This topology comprises multiple domains consisting of routers interconnected by point-to-point data links, LANs, and wide area networks such as ATM.

The principal task of a packet-switching network is to provide connectivity among users. The preceding description of the existing packet-switching network infrastructure reveals the magnitude of this task. Routers exchange information among themselves and use routing protocols to build a consistent set of routing tables that can be used in the routes to direct the traffic flows in these networks. The routing protocols must adapt to changes in network topology due to the introduction of new nodes and links or to failures in equipment. Different routing algorithms are used within a domain and between domains. A key concern here is that the routing tables result in stable traffic flows that make efficient use of network resources. Another concern is to keep the size of routing tables manageable even as the size of the network continues to grow at a rapid pace. In this chapter we show how hierarchical addressing structures can help address this problem. A third concern is to deal with congestion that inevitably occurs in the network. It makes no sense to accept packets into the network when they are likely to be discarded. Thus when congestion occurs inside the network, that is, buffers begin filling up as a result of a surge in traffic or a fault in equipment, the network should react by applying congestion control to limit access to the network only to traffic that is likely to be delivered. A final concern involves providing the capability to offer Quality-of-Service guarantees to some packet flows. We deal with these topics also in the remainder of the chapter.

7.3 DATAGRAMS AND VIRTUAL CIRCUITS

A network is usually represented as a cloud with multiple input sources and output destinations as shown in Figure 7.9. A network can be viewed as a generalization of a physical cable in the sense of providing connectivity between multiple users. Unlike a cable, a switched network is geographically distributed and consists of a graph of transmission lines (i.e., links) interconnected by switches (nodes). These transmission and switching resources are configured to enable the flow of information among users.

[1]The dynamic host configuration protocol (DHCP) provides users with temporary IP addresses and is discussed in Chapter 8.

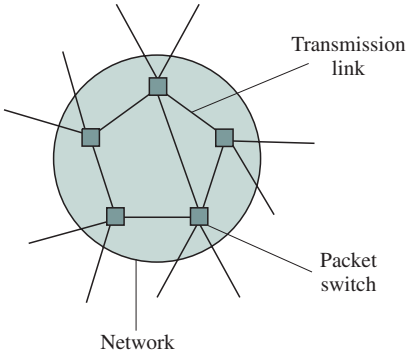


FIGURE 7.9 Switched network

Networks provide for the interconnection of sources to destinations on a dynamic basis. Resources are typically allocated to an information flow only when needed. In this manner the resources are shared among the community of users resulting in efficiency and lower costs. There are two fundamental approaches to transferring information over a packet-switched network. The first approach, called **connection-oriented**, involves setting up a connection across the network before information can be transferred. The setup procedure typically involves the exchange of signaling messages and the allocation of resources along the path from the input to the output for the duration of the connection. The second approach is **connectionless** and does not involve a prior allocation of resources. Instead a packet of information is routed independently from switch to switch until the packet arrives at its destination. Both approaches involve the use of switches or routers to direct packets across the network.

7.3.1 Structure of Switch/Router

Figure 7.10 shows a generic switch consisting of input ports, output ports, an interconnection fabric, and a switch controller/processor. Input ports and output ports are usually paired. A line card typically handles several input/output ports. The line card implements physical and data link layer functions. Thus the card is concerned with symbol timing and line coding. It is also concerned with framing, physical layer addressing, and error checking. For widely deployed standards, the line card also implements medium access control and data link protocols in hardware with a special-purpose chip set. The line card also contains some buffering to handle the speed mismatch between the transmission line and the interconnection fabric. The controller/processor can carry out a number of functions depending on the type of packet switching. The function of the interconnection fabric is to transfer packets between the line cards. Note that Figure 7.10 shows an "unfolded" version of the switch in which the line cards appear twice, once with input ports and again with output ports. In the actual implementation the transmit and receive functions take place in a single line card. However, the function of various types of switch architectures is easier to visualize this way.

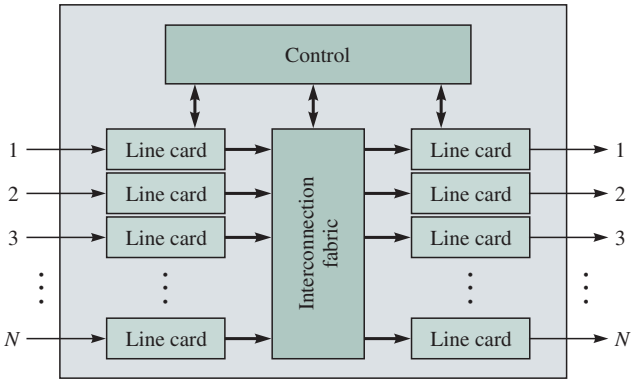


FIGURE 7.10 Components of generic switch/router

We elaborate on the operation of the switches as we develop the two approaches to packet switching.

A simple switch can be built by using a personal computer or a workstation and inserting several network interface cards (NICs) in the expansion slots as shown in Figure 7.11. The frames that arrive at the NICs are de-encapsulated, and the packets are transferred by using the I/O bus from the NIC to main memory. The processor performs the required routing and protocol processing, formats the packet header, and then forwards the packet by transferring it from main memory to the appropriate NIC.

The simple setup in Figure 7.11 reveals the three basic resources and potential bottlenecks in switches: processing, memory, and bus (interconnection) bandwidth. Processing is required to implement the protocols, and hence the processing capacity places a limit on the maximum rate at which the switch can operate. Memory is required to store packets, and hence the amount of memory available determines the rate at which packets are lost, thus placing another limit on the load at which the switch can be operated. In this approach

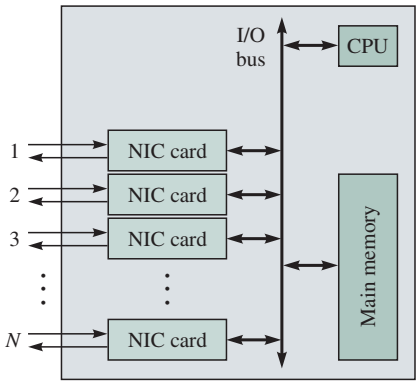


FIGURE 7.11 Building a switch from a general purpose computer

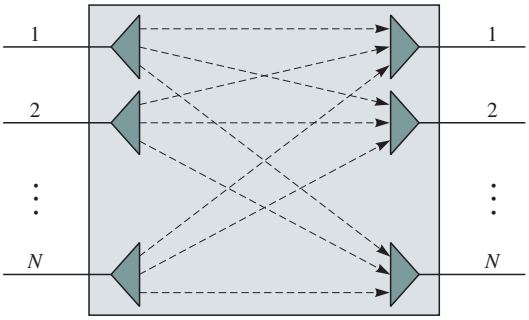


FIGURE 7.12 Input port demultiplexes incoming packet stream; packets are routed to output port; output port multiplexes outgoing packet stream

the memory bandwidth, which is the rate at which information can be read in and out of RAM, also places a limit on the aggregate rate of the switch. Finally, the I/O bus bandwidth places a limit on the total rate at which information can be transferred between ports. Different switch architectures configure these basic resources so that target aggregate switch capacities are met in a cost-effective manner.

Each input and output port in a switch/router typically contains multiplexed streams of packets. Figure 7.12 shows that the flows that enter the switch in effect are demultiplexed at the input port. The switch or router then directs the packets to output ports. Each output port can be viewed as a multiplexer that precedes the outgoing transmission line. Thus we see that switches and routers play a key role in controlling where the packet flows are placed in a network. By controlling packet flows, the network bandwidth can be used efficiently and the performacne can be optimized. We return to this discussion when we discuss Quality-of-Service mechanisms later in the chapter.

HOW TO MAKE BIG, FAST SWITCHES/ROUTERS

Big switches and routers are needed to handle the traffic loads in core networks. An examination of Figure 7.10 shows that the controller and the interconnection fabric are likely to be the bottlenecks. Two strategies can be used to increase switch size. First, as the volume of traffic increases, the placement of a dedicated controller/processor in each line card is justified. This step removes a centralized controller as a potential bottleneck. Second, bus and broadcast type of interconnection structures can be replaced by large bandwidth interconnection fabrics that transfer packets in parallel between input and output ports. A large literature explains how to design switch interconnection fabrics; for example, see [Robertazzi 1994].

7.3.2 Connectionless Packet Switching

Packet switching has its origin in **message switching**, where a message is relayed from one station to another until the message arrives at its destination. At the source each message has a header attached to it to provide source and destination addresses. CRC checkbits are attached to detect errors. As shown in Figure 7.13, the message is transmitted in a store-and-forward fashion. The message is transmitted in its entirety from one switch to the next switch. Each switch performs an error check, and if no errors are found, the switch examines the header to determine the next hop in the path to the destination. If errors are detected, a retransmission may be requested. After the next hop is determined, the message waits for transmission over the corresponding transmission link. Because the transmission links are shared, the message may have to wait until previously queued messages are transmitted. Message switching does not involve a call setup. Message switching can achieve a high utilization of the transmission line. This increased utilization is achieved at the expense of queueing delays. Loss of messages may occur when a switch has insufficient buffering to store the arriving message.[2] End-to-end mechanisms are required to recover from these losses.

Figure 7.14 shows the total delay that is incurred when a message is transmitted over a path that involves two intermediate switches. The message must first traverse the link that connects the source to the first switch. We assume that this link has a propagation delay of p seconds.[3] We also assume that the message has a transmission time of T seconds. The message must next traverse the link connecting the two switches, and then it must traverse the link connecting the second switch and the destination. For simplicity we assume that the

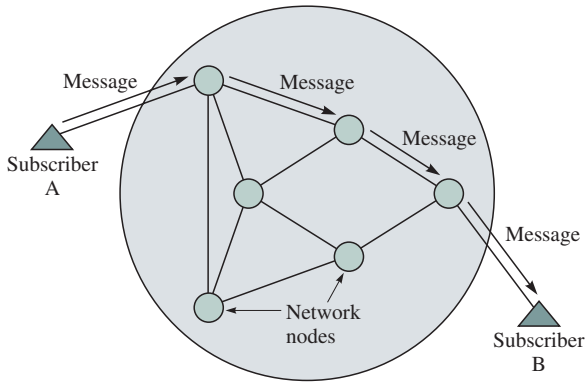


FIGURE 7.13 Message switching

[2]The trade-offs between delay and loss are explored in Chapter 5, section 5.6.1.
[3]The propagation delay is the time that elapses from when a bit enters a transmission line to when it exits the line at the other end.

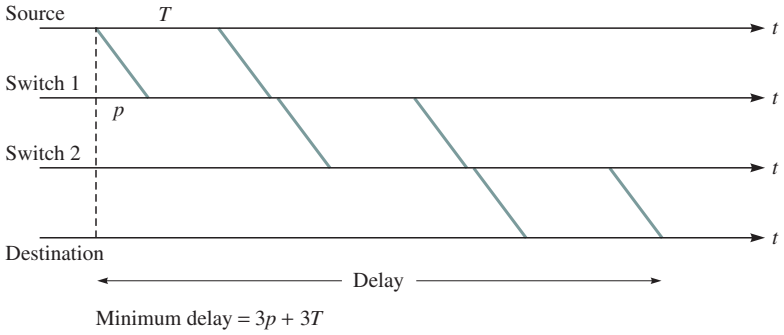


FIGURE 7.14 Delays in message switching

propagation delay and the bit rate of the transmission lines are the same. It then follows that the minimum end-to-end messge delay is $3p + 3T$. Note that this delay does not take into account any queueing delays that may be incurred in the various links waiting for prior messages to be transmitted. It also does not take into account the times required to perform the error checks or any associated retransmissions.

Example—Long Messages versus Packets

Suppose that we wish to transmit a large message ($L = 10^6$ bits) over two hops. Suppose that the transmission line in each hop has an error rate of $p = 10^{-6}$ and that each hop does error checking and retransmission. How many bits need to be transmitted using message switching?

If we transmit the message in its entirety, the probability that the message arrives correctly after the first hop is

$$P_c = (1-p)^L = (1-10^{-6})^{1000000} \approx e^{-Lp} = e^{-1} \approx 1/3$$

Therefore, on the average it will take three tries to get the message over the first hop. Similarly, the second hop will require another three full message transmissions on the average. Thus 6 Mbits will need to be transmitted to get the 1 Mbit message across.

Now suppose that the message is broken up into ten 10^5-bit packets. The probability that a packet arrives correctly after the first hop is

$$P'_c = (1-10^{-6})^{100000} \approx e^{-1/10} \approx 0.90$$

Thus each packet needs to be transmitted $1/0.90 = 1.1$ times on the average. The message gets transmitted over each hop by transmitting an average of 1.1 Mbit. The total number of bits transmitted over the two hops is then 2.2 Mbits.

The preceding example reiterates our observation on ARQ protocols that the probability of error in a transmitted block increases with the length of the block. Thus very long messages are not desirable if the transmission lines are noisy because they lead to a larger rate of message retransmissions. This situation is one reason that it is desirable to place a limit on the maximum size of the blocks that can be transmitted by the network. Thus long messages should be broken into smaller blocks of information, or *packets*.

Message switching is also not suitable for interactive applications because it allows the transmission of very long messages that can impose very long waiting delays on other messages. By placing a maximum length on the size of the blocks that are transmitted, packet switching limits the maximum delay that can be imposed by a single packet on other packets. Thus packet switching is more suitable than message switching for interactive applications.

In the **datagram**, or **connectionless packet-switching** approach, each packet is routed independently through the network. Each packet has an attached header that provides all of the information required to route the packet to its destination. When a packet arrives at a packet switch, the destination address (and possibly other fields) in the header are examined to determine the next hop in the path to the destination. The packet is then placed in a queue to wait until the given transmission line becomes available. By sharing the transmission line among multiple packets, packet switching can achieve high utilization at the expense of packet queueing delays. We note that **routers** in the Internet are packet switches that operate in datagram mode.

Because each packet is routed independently, packets from the same source to the same destination may traverse different paths through the network as shown in Figure 7.15. For example, the routes may change in response to a network fault. Thus packets may arrive out of order, and resequencing may be required at the destination.

Figure 7.16 shows the delay that is incurred by transmitting a message that is broken into three separate packets. Here we assume that the three packets follow

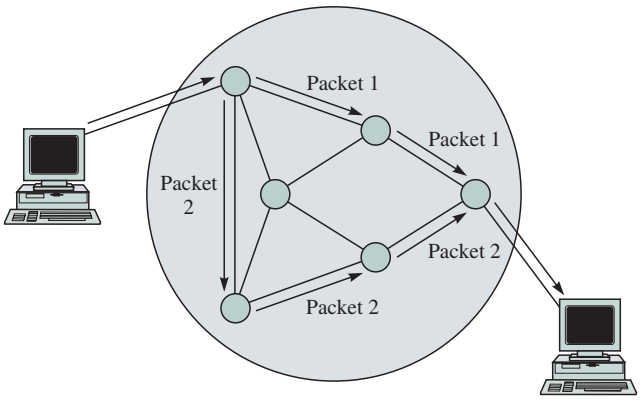


FIGURE 7.15 Datagram packet switching

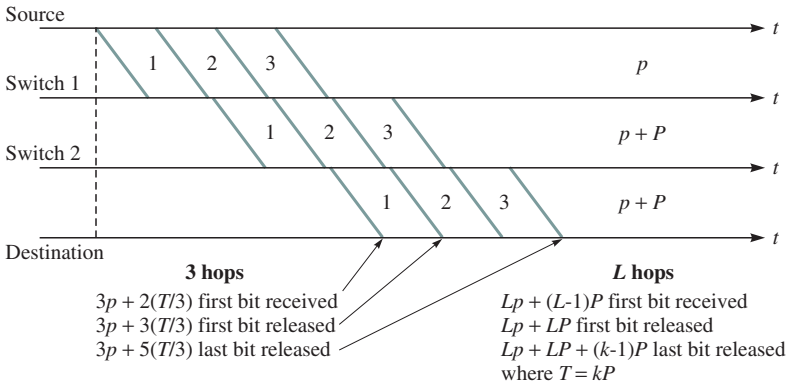


FIGURE 7.16 Delays in packet switching

the same path and are transmitted in succession. We neglect the overhead due to headers and suppose that each packet requires $P = T/3$ seconds to transmit. The three packets are transmitted successively from the source to the first packet switch.

The first packet in Figure 7.16 arrives at the first switch after $p + P$ seconds. Assuming that the packet arrives correctly, it can begin transmission over the next hop after a brief processing time. The first packet is received at the second packet switch at time $2p + 2P$. Again we assume that the packet begins transmission over the final hop after a brief processing time. The first packet then arrives at the final link at time $3p + 3P$. As the first packet traverses the network, the subsequent packets follow immediately, as shown in the figure. In the absence of transmission errors, the final packet will arrive at the destination at time $3p + 3P + 2P = 3p + 5P = 3p + T + 2P$, which is less than the delay incurred in the message switching example in Figure 7.14. In general, if the path followed by a sequence of packets consists of L hops with identical propagation delays and transmission speeds, then the total delay incurred by a message that consists of k packets is given by

$$Lp + LP + (k - 1)P$$

In contrast, the delay incurred using message switching is

$$Lp + LT = Lp + L(kP)$$

Thus message switching involves an additional delay of $(L - 1)(k - 1)P$. We note that the above delays neglect the queueing and processing times at the various hops in the network.

Figure 7.17 shows a routing table that contains an entry for each possible destination for a small network. This entry specifies the next hop that is to be taken by packets with the given destination. When a packet arrives, the destination address in the header is used to perform a table lookup. The result of the lookup is the number of the output port to which the packet must be forwarded.

Destination address	Output port
0785	7
1345	12
1566	6
2458	12

FIGURE 7.17 Routing table in connectionless packet switching

When the size of the network becomes very large, this simple table lookup is not feasible, and the switch/router processor needs to execute a route lookup algorithm for each arriving packet.

In datagram packet switching, the packet switches have no knowledge of a "connection" even when a source and destination exchange a sequence of packets. This feature makes datagram packet switching robust with respect to faults in the network. If a link or packet switch fails, the neighboring packet switches react by routing packets along different links and by sharing the fault information with other switches. This process results in the setting up of a new set of routing tables. Because no connections are set up, the sources and destinations need not be aware of the occurrence of a failure in the network. The processors in the switch/routers execute a distributed algorithm for sharing network state information and for synthesizing routing tables.

The design of the routing table is a key issue in the proper operation of a packet-switching network. This design requires knowledge about the topology of the network as well as of the levels of traffic in various parts of the network. Another issue is that the size of the tables can become very large as the size of the network increases. We discuss these issues further later in the chapter.

Example—IP Internetworks

The Internet Protocol provides for the connectionless transfer of packets across an interconnected set of networks called an internet. In general the component networks may use different protocols so the objective of IP is to provide communications across these dissimilar networks. Each device that is attached to an IP internet has a two-part address: a network part and a host part. To transmit an IP packet, a device sends an IP packet encapsulated using its local network protocol to the nearest router. The routers are packet switches that act as gateways between the component networks. The router performs a route lookup algorithm on the network part of the destination address of the packet to

determine whether the destination is in an immediately accessible network or, if not, to determine the next router in the path to the destination. The router then forwards the IP packet across the given network by encapsulating the IP packet using the format and protocol of the given network. In other words, IP treats the component networks as data link layers whose role is to transfer the packet to the next router or to the destination. IP packets are routed in connectionless fashion from router to router until the destination is reached.

7.3.3 Virtual-Circuit Packet Switching

Virtual-circuit packet switching involves the establishment of a fixed path between a source and a destination prior to the transfer of packets, as shown in Figure 7.18. As in circuit switching, the call setup procedure usually takes place before any packets can flow through the network as shown in Figure 7.19.[4]

The connection setup procedure establishes a path through the network and then sets parameters in the switches along the path as shown in Figure 7.20. The controller/processor in every switch is involved in the exchange of signaling messages to set up the path. As in the datagram approach, the transmission links are shared by packets from many flows. In general, in virtual-circuit packet switching, buffer and transmission resources need not be dedicated explicitly for the use of the connection, but the number of flows admitted may be limited to control the load on certain links. All packets for the connection then follow the same path.

In datagram packet switching each packet must contain the full address of the source and destination. In large networks these addresses can require a large number of bits and result in significant packet overhead and hence wasted transmission bandwidth. One advantage of virtual-circuit packet switching is that

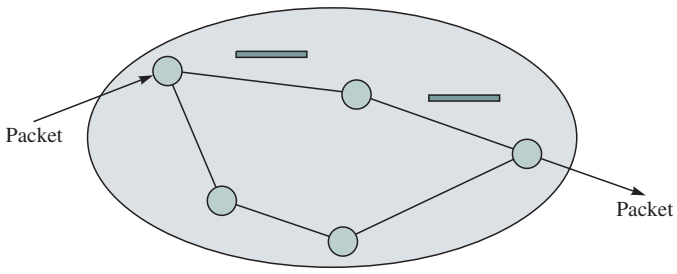


FIGURE 7.18 Virtual-circuit packet switching

[4]In some cases *permanent* virtual circuits are established a priori.

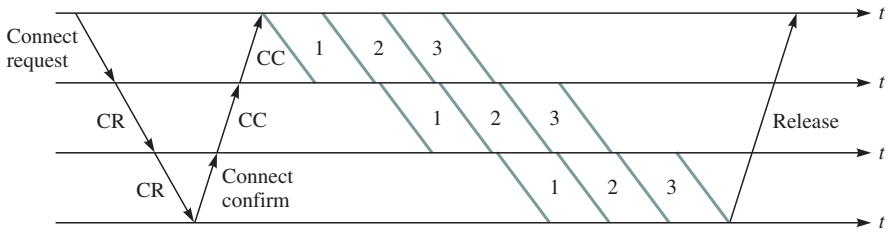


FIGURE 7.19 Delays in virtual-circuit packet switching

abbreviated headers can be used. The connection setup procedure establishes a number of entries in *forwarding tables* located in the various switches along the path. At the input to every switch, the connection is identified by a **virtual-circuit identifier (VCI)**. When a packet arrives at an input port, the VCI in the header is used to access the table, as shown in the example in Figure 7.21. The table lookup provides the output port to which the packet is to be forwarded and the VCI that is to be used at the input port of the next switch. Thus the call setup procedure sets up a chain of pointers across the network that direct the flow of packets in a connection. The table entry for a VCI can also specify the type of priority that is to be given to the packet by the scheduler that controls the transmissions in the next output port.

The number of bits required in the header in virtual-circuit switching is reduced to the number required to represent the maximum number of simultaneous connections over an input port. This number is much smaller than the number required to provide full destination network addresses. This factor is one of the advantages of virtual-circuit switching relative to datagram switching. In addition, the use of abbreviated headers and hardware-based table lookup allows fast processing and forwarding of packets. Virtual-circuit switching does a table lookup and immediately forwards the packet to the output port; connectionless packet switching traditionally was much slower because it required software processing of the header before the next hop in the route could be determined. (This situation has changed with the development of hardware-based routing techniques.)

Another advantage of virtual-circuit packet switching is that resources can be allocated during connection setup. For example, a certain number of buffers may be reserved for a connection at every switch along the path, and a certain amount of bandwidth can be allocated at each link in the path. In addition, the

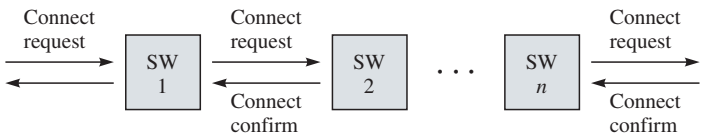


FIGURE 7.20 Signaling message exchanges in virtual-circuit setup

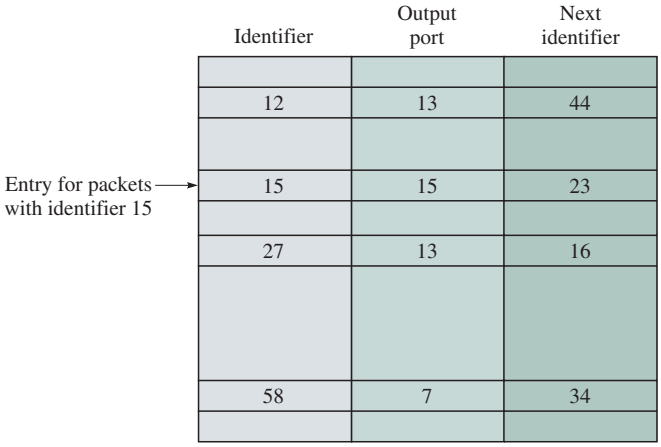


FIGURE 7.21 Example of virtual-circuit routing table for an input port

connection setup process ensures that a switch is able to handle the volume of traffic that is allowed over every transmission link. In particular, a switch may refuse a connection over a certain link when the delays or link utilization exceed certain thresholds.

However, virtual-circuit packet switching does have disadvantages relative to the datagram approach. The switches in the network need to maintain information about the flows they are handling. The amount of required "state" information grows very quickly with the number of flows. Another disadvantage is evident when failures occur. In the case of virtual-circuit packet switching, when a fault occurs in the network all affected connections must be set up again.

If virtual-switching packet switching is used then the minimum delay for transmitting a message that consists of k packets is the same as in Figure 7.16, in addition to the time required to set up the connection. A modified form of virtual-circuit packet switching, called **cut-through packet switching**, can be used when retransmissions are not used in the underlying data link control. It is then possible for a packet to be forwarded as soon as the header is received and the table lookup is carried out. As shown in Figure 7.22, the minimum delay in transmitting the message is then reduced to approximately the sum of the propagation delays in the various hops plus the one-message transmission time. (This scenario assumes that all lines are available to transmit the packet immediately.)

Cut-through packet switching may be desirable for applications such as speech transmission, which has a delay requirement but can tolerate some errors. Cut-through packet switching is also appropriate when the transmission is virtually error free, as in the case of optical fiber transmission, so that hop-by-hop error checking is unnecessary.

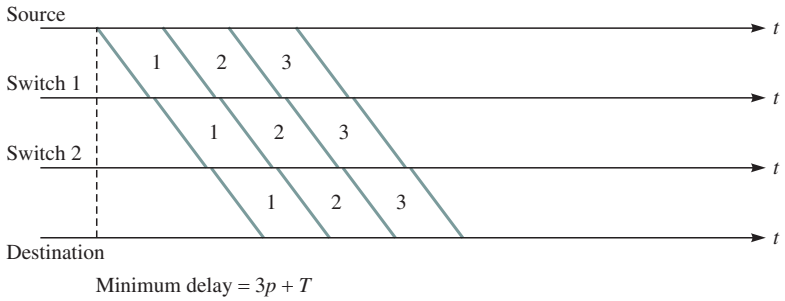


FIGURE 7.22 Cut-through packet switching

Example—ATM Networks

ATM networks provide for the connection-oriented transfer of information across a network. ATM requires all user information to be converted into fixed-length packets called cells. A connection setup phase precedes the transfer of information. During this setup a negotiation takes place in which the user specifies the type of flow that is to be offered to the network, and the network commits to some quality of service that is to be provided to the flow. The connection setup involves setting up a path across the network and allocating appropriate resources along the path.

An ATM connection is defined in terms of a chain of local identifiers called VCIs that identify the connection in each link along the path. Cells are forwarded by ATM switches that perform a table lookup on the VCI to determine the next output port and the VCI in the next link. ATM assumes low-error rate

FLOWS, RESERVATIONS, AND SHORTCUTS

Here we note the emergence of packet-switching approaches that combine features of datagrams and virtual circuits. These hybrid approaches are intended for packet-switching networks that handle a mix of one-time packet transfers (for which datagram mode is appropriate) and sustained packet flows such as long file transfers, Web page downloads, or even steady flows as in audio or video streaming (for which virtual-circuit forwarding is appropriate). In essence these systems attempt to identify longer-term packet flows and to set up shortcuts by using forwarding tables so that packets in a flow are forwarded immediately without the need for route lookup processing. This approach reduces the delay experienced in the packet switch and is discussed further in Chapter 10. Resource reservation procedures for allocating resources to long-term flows have also been developed for datagram networks. We also discuss this in Chapter 10.

optical connections so error control is done only end to end. We discuss ATM in more detail in section 7.6.

7.4 ROUTING IN PACKET NETWORKS

A packet-switched network consists of nodes (routers or switches) interconnected by communication links in an arbitrary meshlike fashion as shown in Figure 7.23. As suggested by the figure, a packet could take several possible paths from host A to host B. For example, three possible paths are 1-3-6, 1-4-5-6, and 1-2-5-6. However, which path is the "best"one? Here the meaning of the term *best* depends on the objective function that the network operator tries to optimize. If the objective is to minimize the number of hops, then path 1-3-6 is the best. If each link incurs a certain delay and the objective function is to minimize the end-to-end delay, then the best path is the one that gives the end-to-end minimum delay. Yet a third objective function involves selecting the path with the greatest available bandwidth. The purpose of the routing algorithm is to identify the set of paths that are best in a sense defined by the network operator. Note that a routing algorithm must have global knowledge about the network state in order to perform its task.

The main ingredients of a good routing algorithm depend on the objective function that one is trying to optimize. However, in general a routing algorithm should seek one or more of the following goals:

1. *Rapid and accurate delivery of packets.* A routing algorithm must operate correctly; that is, it must be able to find a route to the destination if it exists. In addition, the algorithm should not take an unreasonably long time to find the route to the destination.
2. *Adaptability to changes in network topology resulting from node or link failures.* In a real network equipment and transmission lines are subject to failures.

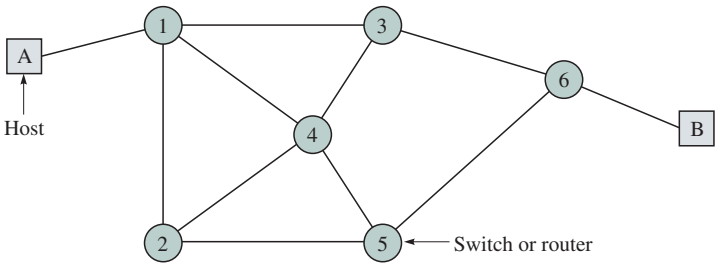


FIGURE 7.23 An example of a packet-switch network

Thus a routing algorithm must be able to adapt to this situation and reconfigure the routes automatically when equipment fails.

3. *Adaptability to varying source-destination traffic loads*. Traffic loads are quantities that are changing dynamically. In a period of 24 hours, traffic loads may go into cycles of heavy and light periods. An adaptive routing algorithm would be able to adjust the routes based on the current traffic loads.
4. *Ability to route packets away from temporarily congested links*. A routing algorithm should try to avoid heavily congested links. Often it is desirable to balance the load on each link.
5. *Ability to determine the connectivity of the network*. To find optimal routes, the routing system needs to know the connectivity or reachability information.
6. *Low overhead*. A routing system typically obtains the connectivity information by exchanging control messages with other routing systems. These messages represent an overhead that should be minimized.

7.4.1 Routing Algorithm Classification

One can classify routing algorithms in several ways. Based on their responsiveness, routing can be static or dynamic (or adaptive). In **static routing** the network topology determines the initial paths. The precomputed paths are then manually loaded to the routing table and remain fixed for a relatively long period of time. Static routing may suffice if the network topology is relatively fixed and the network size is small. Static routing becomes cumbersome as the network size increases. The biggest disadvantage of static routing is its inability to react rapidly to network failures. In **dynamic (adaptive) routing** each router continuously learns the state of the network by communicating with its neighbors. Thus a change in a network topology is eventually propagated to all the routers. Based on the information collected, each router can compute the best paths to desired destinations. One disadvantage of dynamic routing is the added complexity in the router.

Routing algorithms can be centralized or distributed. In **centralized routing** a network control center computes all routing paths and then uploads this information to the routers in the network. In **distributed routing** routers cooperate by means of message exchanges and perform their own routing computations. Distributed routing algorithms generally scale better than centralized algorithms but are more likely to produce inconsistent results. If the routes calculated by different routers are inconsistent, loops can develop. That is, if A thinks that the best route to Z is through B and B thinks that the best route to Z is through A, then packets destined for Z that have the misfortune of arriving at A or B will be stuck in a loop between A and B.

A routing decision can be made on a per packet basis or during the connection setup time. With virtual-circuit packet switching, the path (virtual circuit) is determined during the connection setup phase. Once the virtual circuit is established, all packets belonging to the virtual circuit follow the same route.

Datagram packet switching does not require a connection setup. The route followed by each packet is determined independently.

7.4.2 Routing Tables

Once a routing decision is made, the information has to be stored in a routing table so that the switch (or router) knows how to forward a packet. The specific routing information stored depends on the network type. With virtual-circuit packet switching, the routing table translates each incoming virtual circuit number to an outgoing virtual circuit number and identifies the output port to which to forward the packet. With datagram networks, the routing table identifies the next hop to which to forward the packet based on the destination address of the packet.

Consider a virtual-circuit packet-switching network as shown in Figure 7.24. There are two virtual circuits between host A and switch 1. A packet from host A with VCI 1 in the header will eventually reach host B, while a packet with VCI 5 will eventually reach host D. For each source-destination pair, the VCI has local significance only. At each link the identifier may be translated to a different identifier, depending on the availability of the virtual-circuit numbers at the given switch. In our example VCI 1 from host A gets translated to 2, and then to 7, and finally to 8 at host B. When switch 1 receives a packet with VCI 1, that switch should replace the identifier with 2 and then forward the packet to switch 3. Other switches perform similarly.

Using a local VCI rather than a global one has two advantages. First, more virtual circuits can be assigned, since the virtual-circuit numbers have to be unique only on a link basis rather than on a global basis. If the virtual circuit field in the packet header is two bytes long, then up to 64K virtual circuits can be accommodated on a single link. Second, searching for an available VCI is simple, since a switch has to guarantee uniqueness only on its local link—the information that the switch has in its own routing table. If global virtual-circuit numbers are used, the switch has to make sure that the number if not currently being used by any link along the path, a very time-consuming chore.

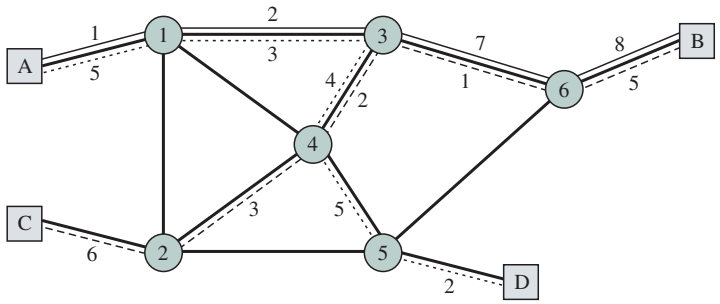


FIGURE 7.24 Virtual-circuit packet switching

The corresponding routing table at each switch is shown in Figure 7.25. We assume that the links are bidirectional and that the VCI is the same for both directions. If a packet with VCI 5 arrives at node 1 from node A, the packet is forwarded to node 3 after the VCI is replaced with 3. After arriving at node 3, the packet receives the outgoing VCI 4 and is then forwarded to node 4. Node 4 translates the VCI to 5 and forwards the packet to node 5. Finally, node 5 translates the VCI to 2 and delivers the packet to the destination, which is host D.

With datagram packet switching, no virtual circuit has to be set up, since no connection exists between a source and a destination. Figure 7.26 shows the routing tables for the network topology in Figure 7.23, assuming that a minimum-hop routing is used. If a packet destined to node 6 arrives at node 1, the packet is first forwarded to node 3 based on the corresponding entry in the routing table at node 1. Node 3 then forwards the packet to node 6. In general, the destination address may be long (32 bits for IPv4), and thus a hash table may be employed to yield a match quickly.

Now suppose that a packet arrives at node 1 from node A and is destined to host D, which is attached to node 5. The routing table in node 1 directs the packet to node 2. The routing table in node 2 directs the packet to node 5, which then delivers the packet to host D.

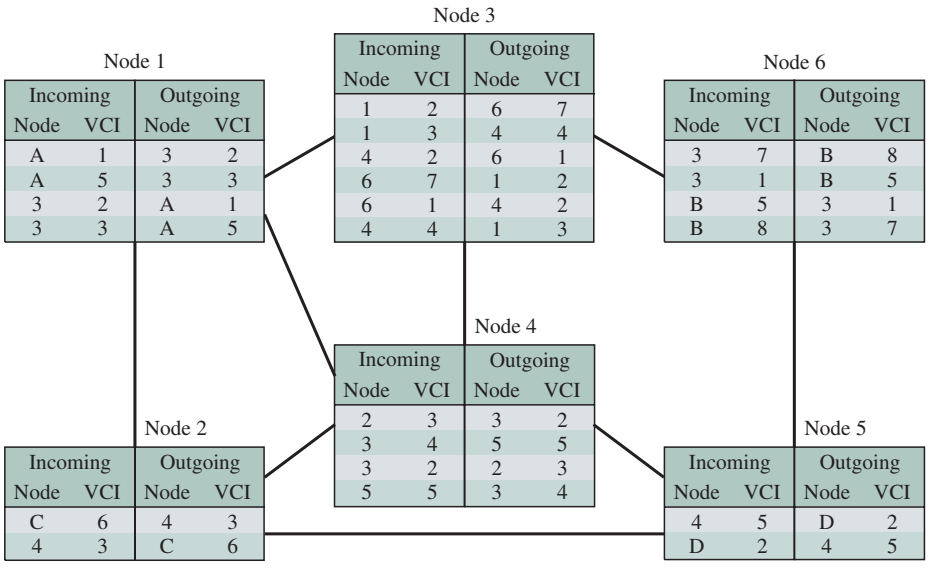


FIGURE 7.25 Routing tables for the network in Figure 7.24

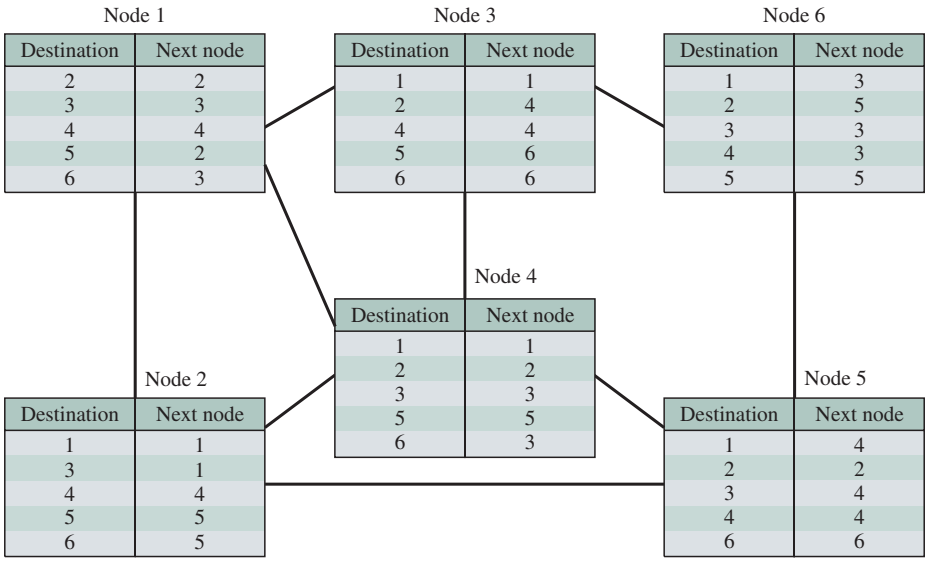


FIGURE 7.26 Routing tables for datagram network

7.4.3 Hierarchical Routing

The size of the routing tables that routers need to keep can be reduced if a hierarchical approach is used in the assignment of addresses. Essentially, hosts that are near each other should have addresses that have common prefixes. In this way routers need to examine only part of the address (i.e., the prefix) in order

HIERARCHICAL ADDRESSES IN THE INTERNET

IP addresses consist of two parts: the first part is a unique identifier for the network within the Internet; the second part identifies the host within the network. IP addresses are made hierarchical in two ways. Within a domain the host part of the address may be further subdivided into two parts: an identifier for a *subnetwork* within the domain and a host identifier within the subnet. Outside the domain, routers route packets according to the network part of the destination address. Once a packet arrives to the domain, further routing is done based on the subnetwork address.

The Internet also uses another hierarchy type for addressing, called *supernetting*. Here networks that connect to a common regional network are given addresses that have a common prefix. This technique allows distant routers to route packets that are destined to networks connected to the same region based on the single routing table entry for the prefix. We explain the details of this procedure when we discuss CIDR addressing in Chapter 8.

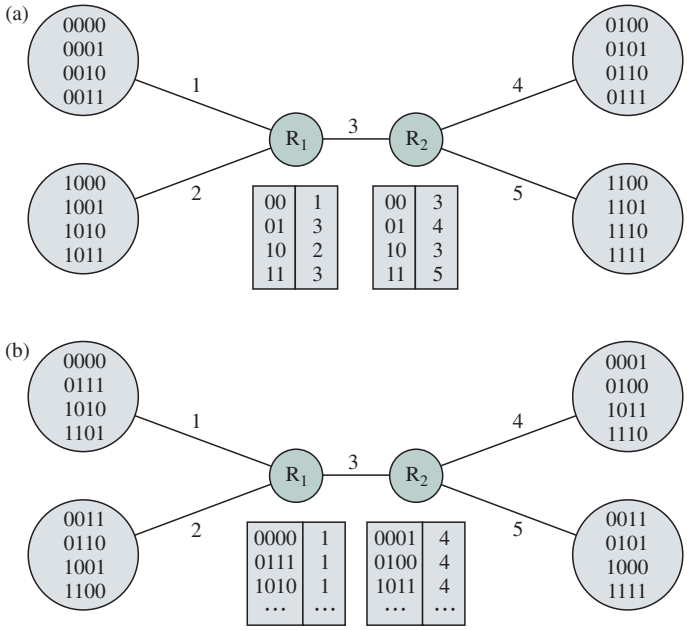


FIGURE 7.27 Hierarchical routing

to decide how a packet should be routed. Figure 7.27 gives an example of hierarchical address assignment and a flat address assignment. In part (a) the hosts at each of the four sites have the same prefix. Thus the two routers need only maintain tables with four entries as shown. On the other hand, if the addresses are not hierarchical (Figure 7.27b), then the routers need to maintain 16 entries in their routing tables.

7.4.4 Link State versus Distance Vector Routing

The set of best paths are invariably found by using a shortest-path algorithm that identifies the set of shortest paths according to some metric. The metric reflects the objective function of the network operator, for example, hops, cost, delay, available bandwidth. To perform the shortest-path calculations, the values of the metrics for different links in the networks are required. Routers must cooperate and exchange information to obtain the values of these metrics. They then use one of the two types of shortest-path algorithms to compute the set of current best routes.

In the *distance vector routing* approach, neighboring routers exchange routing tables that state the set or vector of known distances to other destinations. After neighboring routers exchange this information, they process it to see whether they can find new better routes through the neighbor that provided

UNFINISHED BUSINESS: MULTICASTING

Multicasting involves the delivery of packets to a group of users in a network. Many applications can use multicasting, but the most familiar and suggestive involves receiving the "live" transmission from an audio or video studio. Multicasting has a number of components: addressing to identify multicast groups; mechanisms for joining and leaving a multicast group (i.e., how to "tune in" to a station!); and of course, routing protocols for forwarding the packets from the source to the destinations. Issues in multicasting relating to routing, quality of service, reliability, and security are not completely resolved. We discuss multicast routing in Chapter 8.

the information. Distance vector algorithms adapt to changes in network topology gradually as the information on the changes percolates through the network. In the *link state routing* approach each router floods information about the state of the links that connect it to its neighbors. This action allows each router to construct a map of the entire network and from this map to derive the routing table. Both approaches and the associated algorithms are discussed in section 7.5. The application of these algorithms in Internet routing is discussed in Chapter 8.

7.5 SHORTEST-PATH ALGORITHMS

Network routing is a major component at the network layer and is concerned with the problem of determining feasible paths (or routes) from each source to each destination. A router or a packet-switched node performs two main functions: *routing* and *forwarding*. In the routing function an algorithm finds an optimal path to each destination and stores the result in a routing table. In the forwarding function a router forwards each packet from an input port to the appropriate output port based on the information stored in the routing table. In this section we present two commonly implemented shortest-path routing algorithms: the Bellman-Ford algorithm and Dijkstra's algorithm. We then present several other routing approaches, including flooding, deflection routing, and source routing.

Most routing algorithms are based on variants of **shortest-path algorithms**, which try to determine the shortest path for a packet according to some cost criterion. To better understand the purpose of these algorithms, consider a communication network as a graph consisting of a set of *nodes* (or *vertices*) and a set of links (or *edges*, *arcs*, or *branches*), where each node represents a router or a packet switch and each link represents a communication channel between two routers. Figure 7.28 shows such an example. Associated with each link is a value that represents the *cost* (or *metric*) of using that link. For simplicity, it is assumed

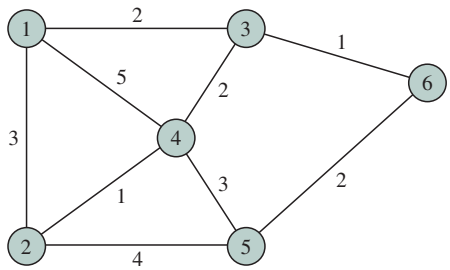


FIGURE 7.28 A sample network with associated link costs

that each link is nondirected. If a link is directed, then the cost must be assigned to each direction. If we define the path cost to be the sum of the link costs along the path, then the shortest path between a pair of nodes is the path with the least cost. For example, the shortest path from node 2 to node 6 is 2-4-3-6, and the path cost is 4.

Many metrics can be used to assign a cost to each link, depending on which function is to be optimized. Examples include

1. *Cost* $\sim$ *1/capacity*. The cost is inversely proportional to the link capacity. Here one assigns higher costs to lower-capacity links. The objective is to send a packet through a path with the highest capacity. If each link has equal capacity, then the shortest path is the path with the minimum number of hops.
2. *Cost* $\sim$ *packet delay*. The cost is proportional to an average packet delay, which includes queueing delay in the switch buffer and propagation delay in the link. The shortest path represents the fastest path to reach the destination.
3. *Cost* $\sim$ *congestion*. The cost is proportional to some congestion measure, for example, traffic loading. Thus the shortest path tries to avoid congested links.

7.5.1 The Bellman-Ford Algorithm

The **Bellman-Ford algorithm** (also called the Ford-Fulkerson algorithm) is based on a principle that is intuitively easy to understand: If a node is in the shortest path between A and B, then the path from the node to A must be the shortest path and the path from the node to B must also be the shortest path. As an example, suppose that we want to find the shortest path from node 2 to node 6 (the destination) in Figure 7.28. To reach the destination, a packet from node 2 must first go through node 1, node 4, or node 5. Suppose that someone tells us that the shortest paths from nodes 1, 4, and 5 to the destination (node 6) are 3, 3, and 2, respectively. If the packet first goes through node 1, *the total distance* (also called total cost) is $3 + 3$, which is equal to 6. Through node 4, the total distance is $1 + 3$, equal to 4. Through node 5, the total distance is $4 + 2$, equal to 6. Thus the shortest path from node 2 to the destination node is achieved if the packet first goes through node 4.

To formalize this idea, let us first fix the destination node. Define D_j to be the current estimate of the minimum cost (or minimum distance) from node j to the destination node and C_{ij} to be the link cost from node i to node j. For example, $C_{12} = C_{21} = 2$, and $C_{45} = 3$ in Figure 7.28. The link cost from node i to itself is defined to be zero (that is, $C_{ii} = 0$), and the link cost between node i and node k is infinite if node i and node k are not directly connected. For example, $C_{15} = C_{23} = \infty$ in Figure 7.28. With all these definitions, the minimum cost from node 2 to the destination node (node 6) can be calculated by

$$\begin{aligned} D_2 &= \min\{C_{21} + D_1, C_{24} + D_4, C_{25} + D_5\} \\ &= \min\{3 + 3, 1 + 3, 4 + 2\} \\ &= 4 \end{aligned} \tag{1}$$

Thus the minimum cost from node 2 to node 6 is equal to 4, and the next node to visit is node 4.

One problem in our calculation of the minimum cost from node 2 to node 6 is that we have assumed that the minimum costs from nodes 1, 4, and 5 to the destination were known. In general, these nodes would not know their minimum costs to the destination without performing similar calculations. So let us apply the same principle to obtain the minimum costs for the other nodes. For example,

$$D_1 = \min\{C_{12} + D_2, C_{13} + D_3, C_{14} + D_4\} \tag{2}$$

and

$$D_4 = \min\{C_{41} + D_1, C_{42} + D_2, C_{43} + D_3, C_{45} + D_5\} \tag{3}$$

A discerning reader will note immediately that these equations are circular, since D_2 depends on D_1 and D_1 depends on D_2. The magic is that if we keep iterating and updating these equations, the algorithm will eventually converge to the correct result. To see this outcome, assume that initially $D_1 = D_2 = \ldots = D_5 = \infty$. Observe that at each iteration, $D_1, D_2, \ldots, D_5$ are nonincreasing. Because the minimum distances are bounded below, eventually $D_1, D_2, \ldots, D_5$ must converge.

Now if we define the destination node, we can summarize the Bellman-Ford algorithm as follows:

1. Initialization

$$\begin{aligned} D_i &= \infty, \forall i \neq d \\ D_d &= 0 \end{aligned} \tag{4}$$

2. Updating: For each $i \neq d$,

$$D_i = \min_j\{C_{ij} + D_j\}, \forall j \neq i \tag{5}$$

Repeat step 2 until no more changes occur in the iteration.

Example—Minimum Cost

Using Figure 7.28, apply the Bellman-Ford algorithm to find both the minimum cost from each node to node 6 (the destination) and the next node along the shortest path.

Let us label each node i by (n, D_i), where n is the next node along the current shortest path and D_i is the current minimum cost from node i to the destination. The next node is found from the value of j in equation 5, which gives the minimum cost. If the next node is not defined, we set n to -1. Table 7.1 shows the execution of the Bellman-Ford algorithm at the end of each iteration. The algorithm terminates after the third iteration, since no more changes are observed. The last row records the minimum cost and the next node along the shortest path from each node to node 6.

Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
Initial	$(-1, \infty)$	$(-1, \infty)$	$(-1, \infty)$	$(-1, \infty)$	$(-1, \infty)$
1	$(-1, \infty)$	$(-1, \infty)$	(6, 1)	(3, 3)	(6, 2)
2	(3, 3)	(4, 4)	(6, 1)	(3, 3)	(6, 2)
3	(3, 3)	(4, 4)	(6, 1)	(3, 3)	(6, 2)

TABLE 7.1 Sample processing of Bellman-Ford algorithm. Each entry for node j represents the next node and cost of the current shortest path to destination 6.

Example—Shortest-Path Tree

From the preceding example, draw the shortest path from each node to the destination node. From the last row of Table 7.1, we see the next node of node 1 is node 3, the next node of node 2 is node 4, the next node of node 3 is node 6, and so forth. Figure 7.29 shows the shortest-path tree rooted at node 6.

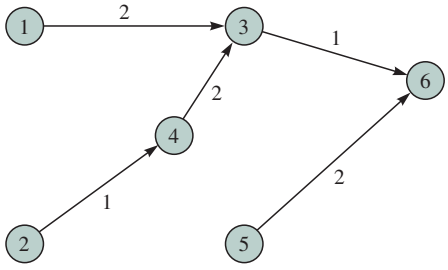


FIGURE 7.29 Shortest-path tree to node 6

One nice feature of the Bellman-Ford algorithm is that it lends itself readily to a distributed implementation. The process involves having each node independently compute its minimum cost to each destination and periodically broadcast the vector of minimum costs to its neighbors. Changes in the routing table

can also trigger a node to broadcast the minimum costs to its neighbors to speed up convergence. This mechanism is called *triggered updates*. It turns out that the distributed algorithm would also converge to the correct minimum costs under mild assumptions. Upon convergence, each node would know the minimum cost to each destination and the corresponding next node along the shortest path. Because only cost vectors (or distance vectors) are exchanged among neighbors, the distributed Bellman-Ford algorithm is often referred to as a *distance vector algorithm*. Each node i participating in the distance vector algorithm computes the following equation:

$$\begin{aligned} D_{ii} &= 0 \\ D_{ij} &= \min_k \{C_{ik} + D_{kj}\}, \forall\ k \neq i \end{aligned} \tag{6}$$

where D_{ij} is the minimum cost from node i to the destination node j. Upon updating, node i broadcasts the vector $\{D_{i1}D_{i2}D_{i3}\ldots\}$ to its neighbors. The distributed version can adapt to changes in link costs or topology as the next example shows.

Example—Recomputing Minimum Cost

Suppose that after the distributed algorithm stabilizes for the network shown in Figure 7.28, the link connecting node 3 and node 6 breaks. Compute the minimum cost from each node to the destination node (node 6), assuming that each node immediately recomputes its cost after detecting changes and broadcasts its routing updates to its neighbors. The new network topology is shown in Figure 7.30.

For simplicity assume that the computation and transmission are synchronous. As soon as node 3 detects that link (3,6) breaks, node 3 recomputes the minimum cost to node 6 and finds that the new minimum cost is 5 via node 4 (as indicated in the first update in Table 7.2). It then sends the new routing update to its neighbors, which are nodes 1 and 4. These nodes then recompute their minimum costs (update 2). Node 1 transmits its routing table to nodes 2, 3, and 4, and node 4 transmits its routing table to nodes 1, 2, 3, and 5. After the messages from nodes 1 and 4 are received, nodes 2 and 3 will update their minimum costs (update 3).

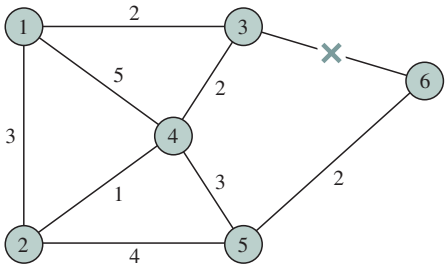


FIGURE 7.30 New network topology following break from node 3 to 6

Update	Node 1	Node 2	Node 3	Node 4	Node 5
Before break	(3, 3)	(4, 4)	(6, 1)	(3, 3)	(6, 2)
1	(3, 3)	(4, 4)	(4, 5)	(3, 3)	(6, 2)
2	(3, 7)	(4, 4)	(4, 5)	(2, 5)	(6, 2)
3	(3, 7)	(4, 6)	(4, 7)	(2, 5)	(6, 2)
4	(2, 9)	(4, 6)	(4, 7)	(5, 5)	(6, 2)
5	(2, 9)	(4, 6)	(4, 7)	(5, 5)	(6, 2)

TABLE 7.2 Next node and cost of current shortest path to node 6

Next nodes 1 and 4 update their minimum costs and send the update messages to their neighbors (update 4). In the last row no more changes are detected, and the algorithm converges. Note that during the calculation, packets already in transit may loop among nodes but eventually find the destination.

Example—Reaction to Link Failure

This example shows that the distributed Bellman–Ford algorithm may react slowly to a link failure. To see this, consider the topology shown in Figure 7.31a with node 4 as the destination. Suppose that after the algorithm stabilizes, link (3,4) breaks, as shown in Figure 7.31b. Recompute the minimum cost from each node to the destination node (node 4).

The computation of minimum costs is shown in Table 7.3. As the table shows, each node keeps updating its cost (in increments of 2 units). At each

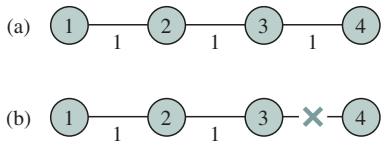


FIGURE 7.31 Topology before and after link failure

Update	Node 1	Node 2	Node 3
Before break	(2, 3)	(3, 2)	(4, 1)
After break	(2, 3)	(3, 2)	(3, 3)
1	(2, 3)	(3, 4)	(3, 3)
2	(2, 5)	(3, 4)	(3, 5)
3	(2, 5)	(3, 6)	(3, 5)
4	(2, 7)	(3, 6)	(3, 7)
5	(2, 7)	(3, 8)	(3, 7)
...	...	...	...

Note: Dots in the last row indicate that the table continues to infinity

TABLE 7.3 Routing table for Figure 7.31

iteration, node 2 thinks that the shortest path to the destination is through node 3. Likewise, node 3 thinks the best path is through node 2. As a result, a packet in either of these two nodes bounces back and forth until the algorithm stops updating. Unfortunately, in this case the algorithm keeps iterating until the minimum cost is infinite (or very large, in practice), at which point, the algorithm realizes that the destination node is unreachable. This problem is often called **counting to infinity**. It is easy to see that if link (3,4) is restored, the algorithm will converge very quickly. Therefore: Good news travels quickly, bad news travels slowly.

To avoid the counting-to-infinity problem, several changes to the algorithm have been proposed, but unfortunately, none of them work satisfactorily in all situations. One particular method that is widely implemented is called the **split horizon**, whereby the minimum cost to a given destination is not sent to a neighbor if the neighbor is the next node along the shortest path. For example, if node X thinks that the best route to node Y is via node Z, then node X should not send the corresponding minimum cost to node Z. Another variation called **split horizon with poisoned reverse** allows a node to send the minimum costs to all its neighbors; however, the minimum cost to a given destination is set to infinity if the neighbor is the next node along the shortest path. Here, if node X thinks that the best route to node Y is via node Z, then node X should set the corresponding minimum cost to infinity before sending it to node Z.

Example—Split Horizon with Poisoned Reverse

Consider again the topology shown in Figure 7.31a. Suppose that after the alogirthm stabilizes, link (3,4) breaks. Recompute the minimum cost from each node to the destination node (node 4), using the split horizon with poisoned reverse.

The computation of minimum costs is shown in Table 7.4. After the link breaks, node 3 sets the cost to the destination equal to infinity, since the minimum cost node 3 has received from node 2 is also infinity. When node 2 receives the update message, it also sets the cost to infinity. Next node 1 also learns that the destination is unreachable. Thus split horizon with poisoned reverse speeds up convergence in this case.

Update	Node 1	Node 2	Node 3
Before break	(2, 3)	(3, 2)	(4, 1)
After break	(2, 3)	(3, 2)	$(-1, \infty)$
1	(2, 3)	$(-1, \infty)$	$(-1, \infty)$
2	$(-1, \infty)$	$(-1, \infty)$	$(-1, \infty)$

TABLE 7.4 Minimum costs by using split horizon with poisoned reverse

---

7.5.2 Dijkstra's Algorithm

Dijkstra's algorithm is an alternative algorithm for finding the shortest paths from a source node to all other nodes in a network. It is generally more efficient than the Bellman-Ford algorithm but requires each link cost to be positive, which is fortunately the case in communication networks. The main idea of Dijkstra's algorithm is to keep identifying the closest nodes from the source node in order of increasing path cost. The algorithm is iterative. At the first iteration the algorithm finds the closest node from the source node, which must be the neighbor of the source node if link costs are positive. At the second iteration the algorithm finds the second-closest node from the source node. This node must be the neighbor of either the source node or the closest node to the source node; otherwise, there is a closer node. At the third iteration the third-closest node must be the neighbor of the first two closest nodes, and so on. Thus at the kth iteration, the algorithm will have determined the k closest nodes from the source node.

The algorithm can be implemented by maintaining a set N of *permanently labeled nodes*, which consists of those nodes whose shortest paths have been determined. At each iteration the next-closest node is added to the set until all nodes are used. To formalize the algorithm, let us define D_i to be the current minimum cost from the source node (labeled s) to node i. Dijkstra's algorithm can be described as follows:

1. Initialization:

$$\begin{aligned} N &= \{s\} \\ D_j &= C_{sj}, \forall\, j \neq s \\ D_s &= 0 \end{aligned} \tag{7}$$

2. Finding the next closest node: Find node $i \notin N$ such that

$$D_i = \min_{j \notin N} D_j \tag{8}$$

   Add i to N.
   If N contains all the nodes, stop.

3. Updating minimum costs: For each node $j \notin N$

$$D_j = \min\{D_j, D_i + C_{ij}\} \tag{9}$$

   Go to step 2.

Example—Finding the Shortest Path

Using Figure 7.28, apply Dijkstra's algorithm to find the shortest paths from the source node (assumed to be node 1) to other nodes.

Table 7.5 shows the execution of Dijkstra's algorithm at the end of the initialization and each iteration. At each iteration the value of the minimum cost of the next closest node is underlined. In case of a tie, the closest node

Iteration	N	D_2	D_3	D_4	D_5	D_6
Initial	{1}	3	2	5	∞	∞
1	{1,3}	3	2	4	∞	3
2	{1,2,3}	3	2	4	7	3
3	{1,2,3,6}	3	2	4	5	3
4	{1,2,3,4,6}	3	2	4	5	3
5	{1,2,3,4,5,6}	3	2	4	5	3

TABLE 7.5 Execution of Dijkstra's algorithm

can be chosen randomly. The minimum cost for each node not permanently labeled is then updated sequentially. The last row records the minimum cost to each node.

If we also keep track of the predecessor node of the next-closest node at each iteration, we can obtain a shortest-path tree rooted at node 1, such as shown in Figure 7.32. When the algorithm stops, is knows the minimum cost to each node and the next node along the shortest path. For a datagram network, the routing table at node 1 looks like Table 7.6.

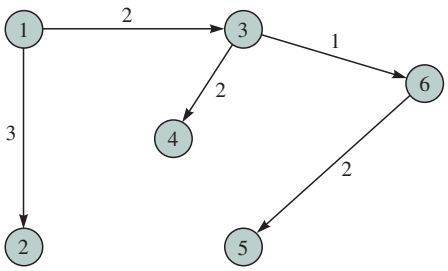


FIGURE 7.32 Shortest-path tree from node 1 to other nodes

Destination	Next node	Cost
2	2	3
3	3	2
4	3	4
5	3	5
6	3	3

TABLE 7.6 Routing table for Figure 7.32

To calculate the shortest paths, the Dijkstra algorithm requires the costs of all links to be available to the algorithm. Thus these link values must be communicated to the processor that is carrying out the computation. The class of *link state* routing algorithms uses this approach to calculating shortest paths. We

discuss these algorithms further in Chapter 8 when we consider how the Internet handles routing.

7.5.3 Other Routing Approaches

Various other routing techniques may be used for other purposes. In this section we look at three common approaches: flooding, deflection routing, and source routing.

FLOODING

The principle of **flooding** calls for a packet switch to forward an incoming packet to all ports except the one the packet was received from. If each switch performs this flooding process, the packet will eventually reach the destination as long as at least one path exists between the source and the destination. Flooding is a very effective routing approach when the information in the routing tables is not available, such as during system startup, or when survivability is required, such as in military networks. Flooding is also effective when the source needs to send a packet to all hosts connected to the network (i.e., broadcast delivery). For example, we will see that the link state routing algorithm uses flooding to distribute the link state information.

However, flooding may easily swamp the network as one packet creates multiple packets that in turn create multiples of multiple packets, generating an exponential growth rate as illustrated in Figure 7.33. Initially one packet arriving at node 1 triggers three packets to nodes 2, 3, and 4. In the second phase nodes 2, 3, and 4 send two, two, and three packets, respectively. These packets arrive at nodes 2 through 6. In the third phase 15 more packets are generated, giving a total of 25 packets after three phases. Clearly, flooding needs to be controlled so that packets are not generated excessively. To limit such a behavior, one can implement a number of mechanisms.

One simple method is to use a time-to-live field in each packet. When the source sends a packet, the time-to-live field is initially set to some small number (say, 10 or smaller). Each switch decrements the field by one before flooding the packet. If the value reaches zero, the switch discards the packet. To avoid unnecessary waste of bandwidth, the time-to-live should ideally be set to the minimum hop number between two furthest nodes (called the diameter of the network). In Figure 7.33 the diameter of the network is two. To have a packet reach any destination, it is sufficient to set the time-to-live field to two.

In the second method, each switch adds its identifier to the header of the packet before it floods the packet. When a switch encounters a packet that contains the identifier of the switch, it discards the packet. This method effectively prevents a packet from going around a loop.

The third method is similar to the second method in that they both try to discard old packets. The only difference lies in the implementation. Here each packet from a given source is identified with a unique sequence number. When a switch receives a packet, the switch records the source address and the sequence

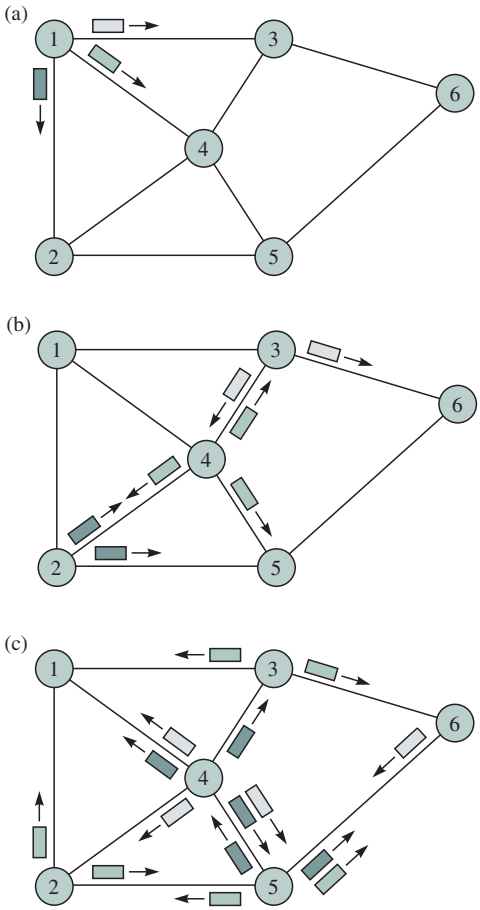


FIGURE 7.33 Flooding is initiated from node 1

number of the packet. If the switch discovers that the packet has already visited the switch, based on the stored source address and sequence number, it will discard the packet.

DEFLECTION ROUTING

Deflection routing was first proposed by Paul Baran in 1964 under the name of *hot-potato routing*. To work effectively, this approach requires the network to provide multiple paths for each source-destination pair. Each switch first tries to forward a packet to the preferred port. If the preferred port is busy or congested, the packet is deflected to another port. Deflection routing usually works well in a regular topology. One example of a regular topology is shown in Figure 7.34, which is called the **Manhattan street network**, since it resembles the streets of New York City. Each column represents an avenue, and each row represents a street. Each switch is labeled (i, j) where i denotes the row number and j denotes the column number. The links have directions that alternate for each column or row. If switch (0,2) would like to send a packet to switch (1,0), the packet could go

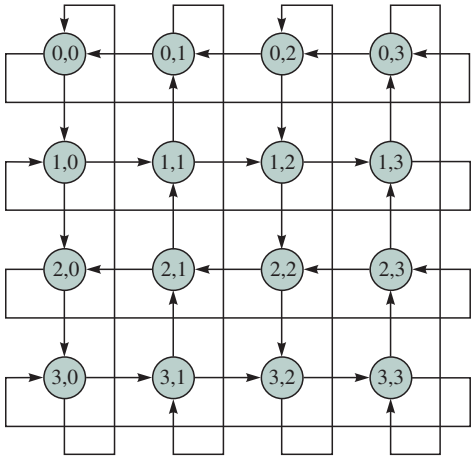


FIGURE 7.34 Manhattan street network

two left and one down. However, if the left port of switch (0,1) is busy (see Figure 7.35), the packet will be deflected to switch (3,1). Then it can go through switches (2,1), (1,1), (1,2), (1,3) and eventually reach the destination switch (1,0).

One advantage of deflection routing is that the switch can be bufferless, since packets do not have to wait for a specific port to become available. If the preferred port is unavailable, the packet can be deflected to another port, which will eventually find its own way to the destination. Since packets can take alternative paths, deflection routing cannot guarantee in-sequence delivery of packets. Deflection routing is a very strong candidate in optical networks where optical buffers are currently expensive and difficult to build. Deflection routing is also used to implement many high-speed packet switches where the topology is usually very regular and high-speed buffers are relatively expensive compared to deflection routing logic.

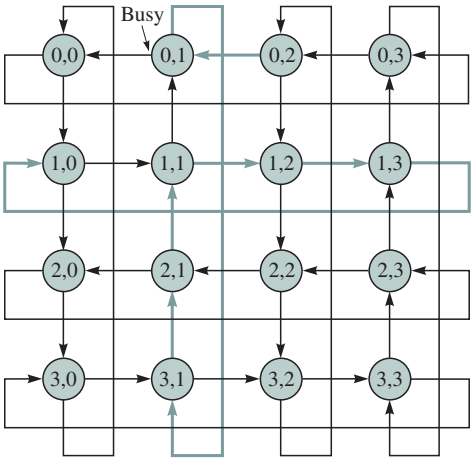


FIGURE 7.35 Deflection routing in Manhattan street network

SOURCE ROUTING

Source routing is a routing approach that does not require an intermediate node to maintain a routing table, but rather puts more burden at the source hosts. Source routing works in either datagram or virtual-circuit packet switching. Before a source host can send a packet, the host has to know the complete route to the destination host in order to include the route information in the header of the packet. The route information contains the sequence of nodes to traverse and should give the intermediate node sufficient information to forward the packet to the next node until the packet reaches the destination. Figure 7.36 shows how source routing works.

Each node examines the header, strips off the label identifying the node, and forwards the packet to the next node. The source (host A) initially includes the entire route (1,3,6,B) in the packet to be destined to host B. Switch 1 strips off its label and forwards the packet to switch 3. The route specified in the header now contains 3,6,B. Switch 3 and switch 6 perform the same function until the packet reaches host B, which finally verifies that it is the intended destination.

In certain situations it may be useful to preserve the complete route information while the packet is progressing. With complete route information host B can send a packet back to host A by simply reversing the route. Thus host B does not have to learn the route to host A a priori. Route preservation can easily be implemented by introducing another field in the header that keeps track of the next node to be visited in the route so that a node knows which specific label to read.

The current version of Internet Protocol, IPv4, and the next version, IPv6, provide an option for source routing of IP packets.

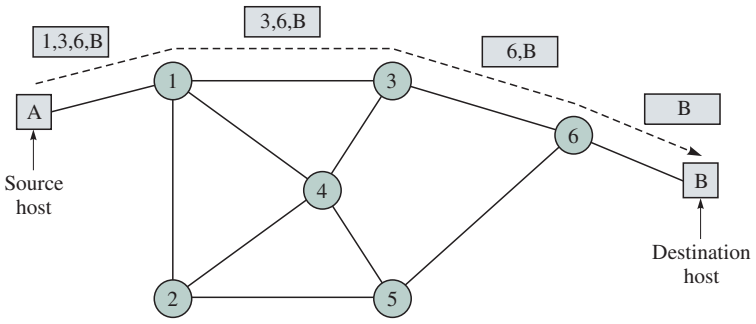


FIGURE 7.36 Example of source routing

7.6 ATM NETWORKS

Asynchronous transfer mode (ATM) is a method for multiplexing and switching that supports a broad range of services. ATM is a connection-oriented packet-switching technique that generalizes the notion of a virtual connection to one that provides Quality-of-Service guarantees.

ATM combines several desirable features of packet switching and time-division multiplexing (TDM) circuit switching. Table 7.7 compares four features of TDM and packet multiplexing. The first comparison involves the capability to support services that generate information at a variable bit rate. Packet multiplexing easily handles variable bit rates. Because the information generated by the service is simply inserted into packets, the variable-bit-rate nature of the service translates into the generation of the corresponding packets. Variable-bit-rate services can therefore be accommodated as long as packets are not generated at a rate that exceeds the speed of the transmission line. TDM systems, on the other hand, have significantly difficulty supporting variable-bit-rate services. Because TDM systems transfer information at a constant bit rate, the bit rates that TDM can support are multiples of some basic rate, for example, 64 kbps for telephone networks.

The second comparison involves the delay incurred in traversing the network. In the case of TDM, once a connection is set up the delays are small and nearly constant. Packet multiplexing, on the other hand, has inherently variable transfer delays because of the queueing that takes place in the multiplexers. Packet multiplexing also has difficulty in providing particular services with low delay. For example, because packets can be of variable length, when a long packet is undergoing transmission, all other packets including urgent ones must wait for the duration of the transmission.

The third criterion for comparison is the capability to support bursty traffic. TDM dedicates the transmission resources, namely, slots, to a connection. If the connection is generating information in a bursty fashion, then many of the dedicated slots go unused. Therefore, TDM is inefficient for services that generate bursty information. Packet multiplexing, on the other hand, was developed specifically to handle bursty traffic and can do so in an efficient way.

Processing is the fourth comparison criterion. In TDM, hardware handles the transfer of slots, so the processing is minimal and can be done at very high speeds. Packet multiplexing, on the other hand, traditionally uses software to

	Variable bit rate	Delay	Bursty traffic	Processing
TDM	Multirate only	Low, fixed	Inefficient	Minimal, very high speed
Packet	Easily handled	Variable	Efficient	Header and packet processing required

TABLE 7.7 TDM versus packet multiplexing

process the information in the packet headers. Consequently, at the time that ATM was formulated, packet multiplexing was slow and processing intensive. However, the development of hardware techniques has reduced packet-processing times and has made very high speed, variable-length packet systems viable.

ATM was developed in the mid-1980s to combine the advantages of TDM and packet multiplexing. ATM involves the conversion of all information flows into short fixed-length packets called **cells**. Cells contain abbreviated headers, or labels, which are essentially pointers to tables in the switches. In terms of the four criteria. ATM has the following features. Because it is packet based, ATM can easily handle services that generate information in bursty fashion or at variable bit rates. The abbreviated header of ATM and the fixed length facilitate hardware implementations that result in low delay and high speeds.

Figure 7.37 shows the operation of an ATM multiplexer. The information flows generated by various users are converted into cells and sent to an ATM multiplexer. The multiplexer arranges the cells into one or more queues and implements some scheduling strategy that determines the order in which cells are transmitted. The purpose of the scheduling strategy is to provide for the different qualities of service required by the different flows. ATM does not reserve transmission slots for specific information flows, and so it has the efficiencies of packet multiplexing. The reason for the term *asynchronous* is that the transmission of cells is not synchronized to any frame structure as in the case of TDM systems.

ATM networks are connection-oriented and require a connection setup prior to the transfer of cells. The connection setup is similar to that described for virtual-circuit packet-switched networks. The connection setup procedure requires the source to provide a *traffic descriptor* that describes the manner in which cells are produced, for example, peak cell rate in cells/second, sustainable (long-term average) cell rate in cells/second, and maximum length of a burst of cells. The source also specifies a set of Quality-of-Service (QoS) parameters that

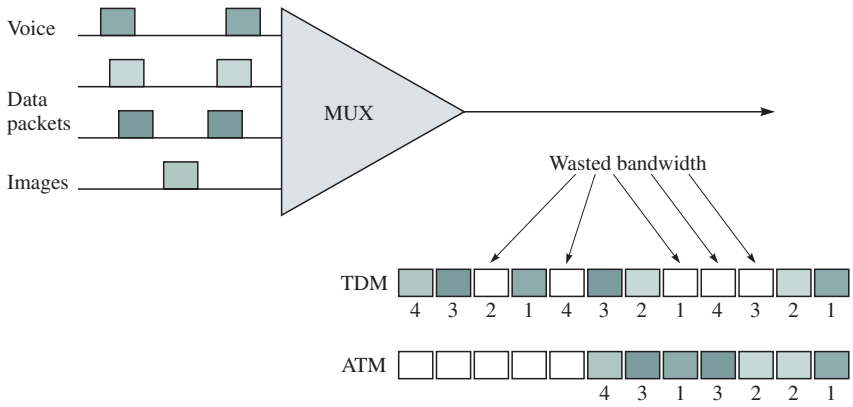


FIGURE 7.37 ATM multiplexing

the connection must provide, for example, cell delay, cell loss, and cell delay jitter. The connection setup procedure involves identifying a path through the network that can meet these requirements. A connection admission control procedure is carried out at every multiplexer along the path. This path is called a **virtual channel connection (VCC)**.

The VCC is established by a chain of *local identifiers* that are defined during the connection setup at the input port to each switch between the source and the destination. The generic packet-switch structure in Figure 7.10 can be modified to carry out ATM switching. Figure 7.38 shows the tables associated with two of the input ports to an ATM switch. In input port 5 we have cells from a voice stream arriving with identifier 32 in the header. We also have cells from a video stream arriving with identifier 25. When a cell with identifier 32 arrives at input port 5, the table lookup for entry 32 indicates that the cell is to be switched to output port 1 and that the identifier in the header is to be changed to 67. Similarly, cells arriving at port 5 with identifier 25 are switched to output port *N* with new identifier 75. Note that the identifier is locally defined for each input port. Thus input port 6 uses identifier 32 for a different VCC.

At this point it is clear that ATM has a strong resemblance to virtual-circuit packet switching. One major difference is ATM's use of short, fixed-length packets. This approach simplifies the implementation of switches and makes very high speed operation possible. Indeed, ATM switches have proven to be scalable to very large sizes, that is, switches of 10,000 ports with each port running at 150 Mbps are possible. The use of short fixed-length packets also gives a finer degree of control over the scheduling of packet transmissions, since the shorter packets imply a smaller minimum waiting time until the transmission line becomes available for the next transmission.

To understand how the local identifiers are defined in ATM, we first need to see how ATM incorporates some of the concepts used in SONET. SONET allows flows that have a common path through the network to be grouped together. ATM uses the concept of a **virtual path** to achieve this bundling. Figure 7.39 shows five VCCs in an ATM network. The VCCs a, b, and c enter

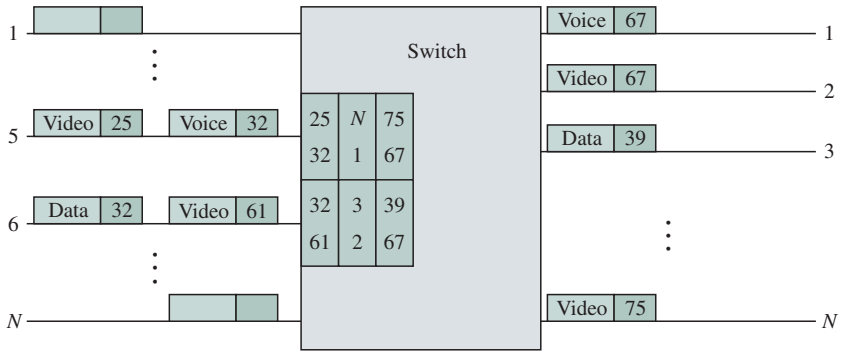


FIGURE 7.38 ATM switching

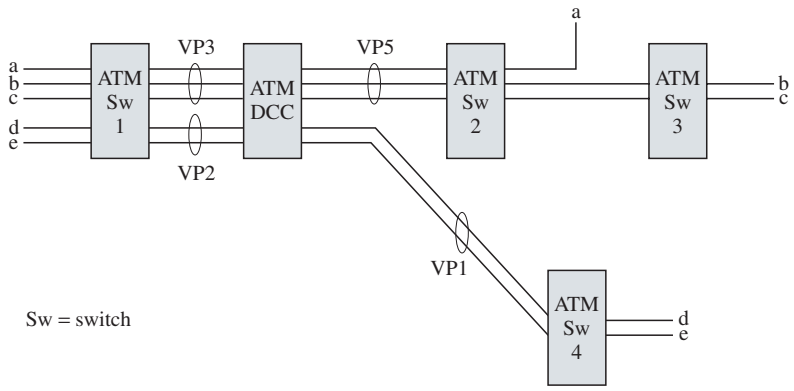


FIGURE 7.39 Role of virtual paths in an ATM network

the network at switch 1, share a common path up to switch 2, and are bundled together into a virtual path connection (VPC) that connects switch 1 to switch 2.[5] This VPC happens to pass through an ATM cross-connect switch whose role in this example is to switch only virtual paths. The VPC that contains VCCs a, b, and c has been given **virtual path identifier (VPI)** 3 between switch 1 and the cross-connect. The cross-connect switches all cells with VPI 3 to the link connecting it to switch number 2 and changes the VPI to 5, which identifies the virtual path between the cross-connect and ATM switch 2. This VPC terminates at switch 2 where the three VCCs are unbundled; cells from VCC a are switched out to a given output port, whereas cells from VCCs b and c proceed to switch 3. Figure 7.39 also shows VCCs d and e entering at switch 1 with a common path to switch 4. These two channels are bundled together in a virtual path that is identified by VPI 2 between switch 1 and the cross-connect and by VPI 1 between the cross-connect and switch 4.

The preceding discussion clearly shows that a virtual circuit in ATM requires two levels of identifiers: an identifier for the VPC, the VPI; and a local identifier for the VCC, the so-called virtual channel identifier, VCI. Figure 7.40 shows a cross-section of the cell stream that arrives at a given input port of an ATM switch or a cross-connect. The cells of a specific VCC are identified by a two-part identifier consisting of a VPI and a VCI. VCCs that have been bundled into a virtual path have the same VPI, and their cells are switched in the same manner over the entire length of the virtual path. At all switches along the virtual path, switching is based on the VPI only and the VCIs are unchanged. The VCIs are used and translated only at the end of the virtual path.

[5]We use letters to identify the end-to-end virtual connection. In ATM the network identifies each virtual connection by a chain of locally defined identifiers.We use numbers to indicate these identifiers.

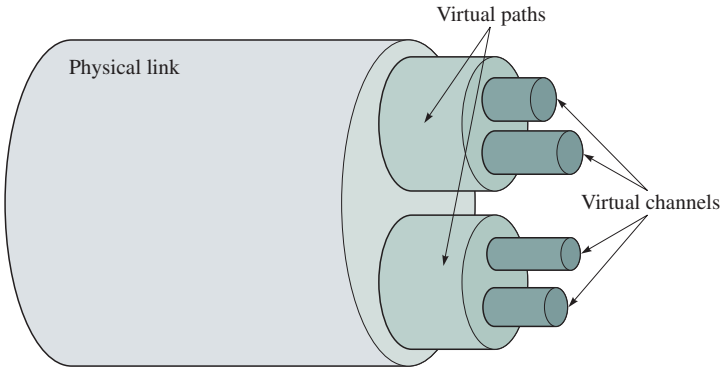


FIGURE 7.40 ATM virtual connections

The details of the VPIs and VCIs are discussed in Chapter 9. However, it is worth noting here that the VCI/VPI structure can support a very large number of connections and hence provides scalability to very large networks.

ATM provides many of the features of SONET systems that facilitate the configuration of the network topology and the management of the bandwidth. The virtual path concept combined with the use of ATM cross-connect switches allows the network operator to dynamically reconfigure the topology seen by the ATM switches using software control. This concept also allows the operator to change the bandwidth allocated to virtual paths. Furthermore, these bandwidth allocations can be done to any degree of granularity, that is, unlike SONET, ATM is not restricted to multiples of 64 kbps.

7.7 Traffic Management and QoS

Traffic management is concerned with the delivery of QoS to specific packet flows. Traffic management entails mechanisms for managing the flows in a network to control the load that is applied to various links and switches. Traffic management also involves the setting of priority and scheduling mechanisms at switches, routers, and multiplexers to provide differentiated treatment for packets and cells belonging to different classes, flows, or connections. It also may involve the policing and shaping of traffic flows as they enter the network.[6]

When we discussed the general structure of switches and routers, we noted in Figure 7.12 that a switch can be viewed as a node where multiplexed packet streams arrive and are then demultiplexed, routed, and remultiplexed onto outgoing lines. Thus the path traversed by a packet through a network can be modeled as a sequence of multiplexers and transmission links as shown in

[6]Traffic management also encompasses congestion control, the topic of the next section.

Figure 7.41. The dashed arrows show packets from other flows that "interfere" with the packet of interest in the sense of contending for buffers and transmission along the path. We also note that these interfering flows may enter at one multiplexer and depart at some later multiplexer, since in general they belong to different source-destination pairs and follow different paths through the network.

The performance experienced by a packet along the path is the accumulation of the performance experienced at the N multiplexers. For example, the total *end-to-end delay* is the sum of the delays experienced at each multiplexer. Therefore, the average end-to-end delay is the sum of the individual average delays. On the other hand, if we can guarantee that the delay at each multiplexer can be kept below some upper bound, then the end-to-end delay can be kept below the sum of the upper bounds at the various multiplexers. The *jitter* experienced by packets is also of interest. The jitter measures the variability in the packet delays and is typically measured in terms of the difference of the minimum delay and some maximum value of delay.

Packet loss performance is also of interest. Packet loss occurs when a packet arrives at a multiplexer that has no more buffers available. Causes of packet loss include surges in packet arrivals to a multiplexer and decreased transmission rates out of a multiplexer due to faults in equipment or congestion downstream. The end-to-end probability of packet loss is the probability of packet loss somewhere along the path and is bounded above by the sum of the packet loss probabilities at each multiplexer.

Note that the discussion here is not limited solely to connection-oriented packet transfer. In the case of connectionless transfer of packets, each packet will experience the performance along the path traversed. If each packet is likely to traverse a different path, then it is difficult to make a statement about packet performance. On the other hand, this analysis will hold in connectionless packet-switching networks for the period of time during which a single path is used between a source and a destination. If these paths can be "pinned down" for certain flows in a connectionless network, then the end-to-end analysis is valid.

Packet-switching networks are called upon to support a wide range of services with diverse QoS requirements. To meet the QoS requirements of multiple services, an ATM or packet multiplexer must implement strategies for managing how cells or packets are placed in the queue or queues, as well as control the transmission bit rates that are provided to the various information flows. We now consider a number of these strategies.

FIGURE 7.41 The end-to-end QoS of a packet along a path traversing N hops

7.7.1 FIFO and Priority Queues

The simplest approach to managing a multiplexer involves **first-in, first-out (FIFO) queueing** where all arriving packets are placed in a common queue and transmitted in order of arrival, as shown in Figure 7.42a. Packets are discarded when they arrive at a full buffer. The delay and loss experienced by packets in a FIFO system depend on the interarrival times and on the packet lengths. As interarrivals become more bursty or packet lengths more variable, performance will deteriorate. Because FIFO queueing treats all packets in the same manner, it is not possible to provide different information flows with different qualities of service. FIFO systems are also subject to *hogging*, which occurs when a user sends packets at a high rate and fills the buffers in the system, thus depriving other users of access to the multiplexer.

A FIFO queueing system can be modified to provide different packet-loss performance to different traffic types. Figure 7.42b shows an example with two classes of traffic. When the number of packets reaches a certain threshold, arrivals of lower access priority (Class 2) are not allowed into the system. Arrivals of higher access priority (Class 1) are allowed as long as the buffer is not full. As a result, packets of lower access priority will experience a higher packet-loss probability.

Head-of-line (HOL) priority queueing is a second approach that involves defining a number of priority classes. A separate queue is maintained for each priority class. As shown in Figure 7.43, each time the transmission line becomes available the next packet for transmission is selected from the head of the line of the highest priority queue that is not empty. For example, packets requiring low delay may be assigned a high priority, whereas packets that are not urgent may be assigned a lower priority. The size of the buffers for the different priority classes can be selected to meet different loss probability requirements. While priority queueing does provide different levels of service to the different classes, it still has shortcomings. For example, it does not allow for providing some

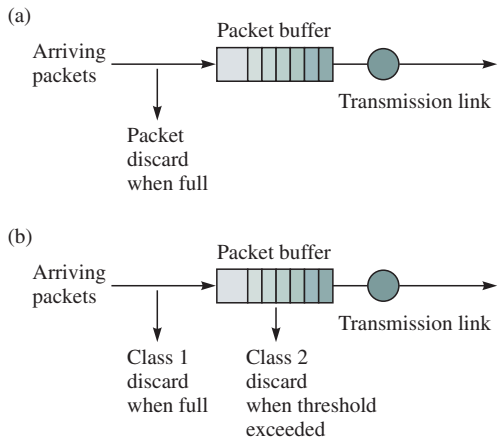


FIGURE 7.42 (a) FIFO queueing; (b) FIFO queueing with discard priority

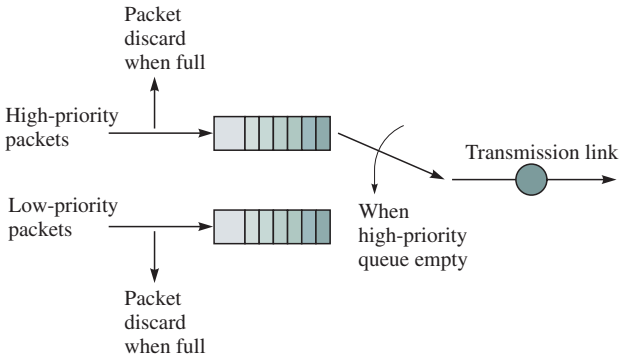


FIGURE 7.43 HOL priority queueing

degree of guaranteed access to transmission bandwidth to the lower priority classes. Another problem is that it does not discriminate between users of the same priority. Fairness problems can arise here when a certain user hogs the bandwidth by sending an excessive number of packets.

A third approach to managing a multiplexer, shown in Figure 7.44, involves sorting packets in the queue according to a priority tag that reflects the urgency with which each packet needs to be transmitted. This system is very flexible because the method for defining priority is open and can even be defined dynamically.[7] For example, the priority tag could consist of a priority class followed by the arrival time of a packet to a multiplexer. The resulting system implements the HOL priority system discussed above. In a second example the priority tag corresponds to a due date. Packets without a delay requirement get indefinite or very long due dates, and are transmitted after all time-critical packets have been transmitted. A third important example that can be implemented by the approach is fair queueing and weighted fair queueing, which are discussed next.

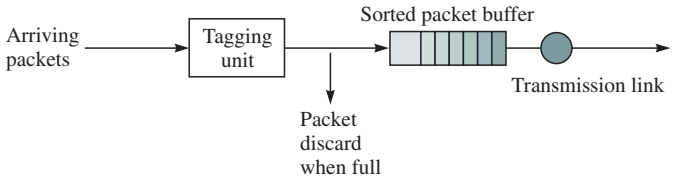


FIGURE 7.44 Sorting packets according to priority tag

[7]See [Hashemi 1997] for a discussion on the various types of scheduling schemes that can be implemented by this approach.

7.7.2 Fair Queueing

Fair queueing attempts to provide equitable access to transmission bandwidth. Each user flow has its own logical queue. In an ideal system the transmission bandwidth, say, C bits/second, is divided equally among the queues that have packets to transmit.[8] The contents of each queue can then be viewed as a *fluid* that is drained continuously. Fair queueing prevents the phenomenon of *hogging*, which occurs when an information flow receives an unfair share of the bit rate. The size of the buffer for each user flow can be selected to meet specific loss probability requirements so that the cells or packets of a given user will be discarded when that buffer is full.

Fair queueing is "fair" in the following sense. In the ideal fluid flow situation, the transmission bandwidth is divided equally among all nonempty queues. Thus if the total number of flows in the system is n and the transmission capacity is C, then each flow is guaranteed at least C/n bits/second. In general, the actual transmission rate experienced may be higher because queues will be empty from time to time, so a share larger than C/n bps is received at those times.

In practice, dividing the transmission capacity exactly equally is not possible. As shown in Figure 7.45 one approach could be to service each nonempty queue one bit at a time in round-robin fashion. However, decomposing the resulting bit stream into the component packets would require the introduction of framing information and extensive processing at the demultiplexer. In the case of ATM, fair queueing can be approximated in a relatively simple way. Because in ATM all packets are the same length, the multiplexer need only service the nonempty queues one packet at a time in round-robin fashion. User flows are then guaranteed equal access to the transmission bandwidth.

Figure 7.46 illustrates the differences between ideal or "fluid flow" and packet-by-packet fair queueing. The figure assumes that queue 1 and queue 2 each has a single L-bit packet to transmit at $t = 0$ and that no subsequent packets arrive. Assuming a capacity of $C = L$ bits/second = 1 packet/second, the fluid-flow system transmits each packet at a rate of 1/2 and therefore

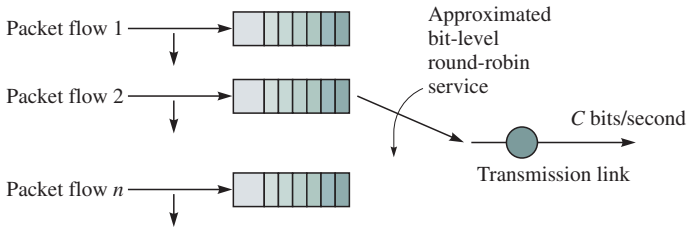


FIGURE 7.45 Fair queueing

[8]This technique is called *processor sharing* in the computing literature.

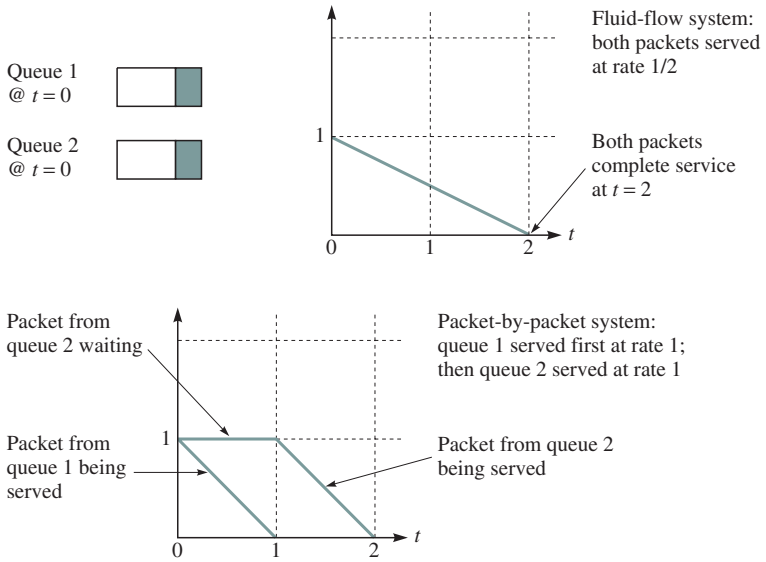


FIGURE 7.46 Fluid-flow and packet-by-packet fair queueing (two packets of equal length)

completes the transmission of both packets exactly at time $t = 2$ seconds. The bit-by-bit system (not shown in the figure) would begin by transmitting one bit from queue 1, followed by one bit from queue 2, and so on. After the first bit each subsequent bit from queue 1 would require $2/L$ seconds to transmit. Therefore, the transmission of the packet from queue 1 would be completed after $1 + 2(L - 1) = 2L - 1$ bit-transmission times, which equals $2 - 1/L$ seconds. The packet from queue 2 is completed at time 2 seconds. The transmission of both packets in the fluid-flow system would be completed at time $2L/L = 2$ seconds. On the other hand, the packet-by-packet fair-queueing system transmits the packet from queue 1 first and then transmits the packet from queue 2, so the packet completion times are 1 and 2 seconds. In this case the first packet is 1 second too early relative to the completion time in the fluid system.

Approximating fluid-flow fair queueing is not as straightforward when packets have variable lengths. If the different user queues are serviced one packet at a time in round-robin fashion, we do not necessarily obtain a fair allocation of transmission bandwidth. For example, if the packets of one flow are twice the size of packets in another flow, then in the long run the first flow will obtain twice the bandwidth of the second flow. A better approach is to transmit packets from the user queues so that the packet completion times approximate those of a fluid-flow fair queueing system. Each time a packet arrives at a user queue, the completion time of the packet is derived from a fluid-flow fair-queueing system. This number is used as a finish tag for the packet. Each time the transmission of a packet is completed, the next packet to be transmitted is the one with the

smallest finish tag among all of the user queues. We refer to this system as a **packet-by-packet fair-queueing** system.

Assume that there are n flows, each with its own queue. Suppose for now that each queue is served one bit at a time. Let a *round* consist of a cycle in which all n queues are offered service as shown in Figure 7.47. The actual duration of a given round is the actual number of queues $n_{active}(t)$ that have information to transmit. When the number of active queues is large, the duration of a round is large; when the number of active queues is small, the rounds are short in duration.

Now suppose that the queues are served as in a fluid-flow system. Also suppose that the system is started at $t = 0$. Let $R(t)$ be the number of the rounds at time t, that is, the number of cycles of service to all n queues. However, we let $R(t)$ be a continuous function that increases at a rate that is inversely proportional to the number of active queues; that is:

$$dR(t)/dt = C/n_{active}(t)$$

where C is the transmission capacity. Note that $R(t)$ is a piecewise linear function that changes in slope each time the number of active queues changes. Each time $R(t)$ reaches a new integer value marks an instant at which all the queues have been given an equal number of opportunities to transmit a bit.

Let us see how we can calculate the finish tags to approximate fluid-flow fair queueing. Suppose the kth packet from flow i arrives at an empty queue at time t_k^i and suppose that the packet has length $P(i, k)$. This packet will complete its transmission when $P(i, k)$ *rounds* have elapsed, one round for each bit in the packet. Therefore, the packet completion time will be the value of time t^* when the $R(t)^*$ reaches the value:

$$F(i, k) = R(t_k^i) + P(i, k)$$

We will use $F(i, k)$ as the **finish tag** of the packet. On the other hand, if the kth packet from the ith flow arrives at a nonempty queue, then the packet will have a finish tag $F(i, k)$ equal to the finish tag of the previous packet in its queue $F(i, k - 1)$ plus its own packet length $P(i, k)$; that is:

$$F(i, k) = F(i, k - 1) + P(i, k)$$

The two preceding equations can be combined into the following compact equation:

$$F(i, k) = \max\{F(i, k - 1), R(t_k^i)\} + P(i, k) \quad \text{for fair queueing.}$$

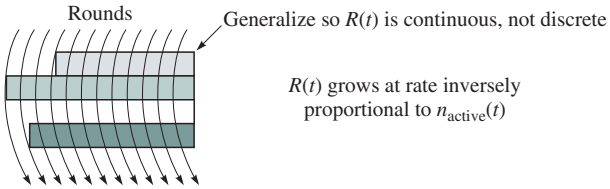


FIGURE 7.47 Computing the finishing time in packet-by-packet fair queueing and weighted fair queueing

We reiterate: The actual packet completion time for the kth packet in flow i in a fluid-flow fair-queueing system is the time t when $R(t)$ reaches the value $F(i, k)$. The relation between the actual completion time and the finish tag is not straightforward because the time required to transmit each bit varies according to the number of active queues.

As an example, suppose that at time $t = 0$ queue 1 has one packet of length one unit and queue 2 has one packet of length two units. A fluid-flow system services each queue at rate 1/2 as long as both queues remain nonempty. As shown in Figure 7.48, queue 1 empties at time $t = 2$. Thereafter queue 2 is served at rate 1 until it empties at time $t = 3$. In the packet-by-packet fair-queueing system, the finish tag of the packet of queue 1 is $F(1, 1) = R(0) + 1 = 1$. The finish tag of the packet from queue 2 is $F(2, 1) = R(0) + 2 = 2$. Since the finish tag of the packet of queue 1 is smaller than the finish tag of queue 2, the system will service queue 1 first. Thus the packet of queue 1 completes its transmissions at time $t = 1$ and the packet of queue 2 completes its transmissions at $t = 3$.

WEIGHTED FAIR QUEUEING

Weighted fair queueing addresses the situation in which different users have different requirements. As before, each user flow has its own queue, but each user flow also has a *weight* that determines its relative share of the bandwidth. Thus if queue 1 has weight 1 and queue 2 has weight 3, then when both queues are nonempty, queue 1 will receive $1/(1 + 3) = 1/4$ of the bandwidth and queue 2 will receive 3/4 of the bandwidth. Figure 7.49 shows the completion times for the

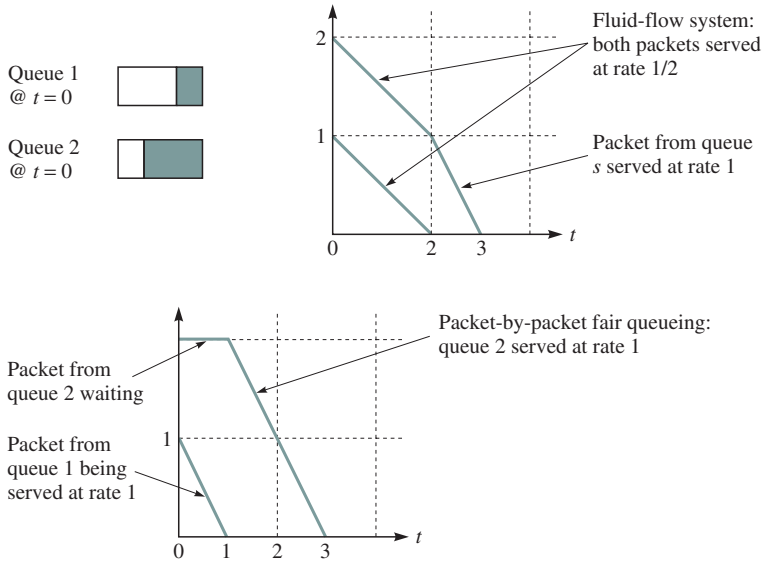


FIGURE 7.48 Fluid flow and packet-by-packet fair queueing (two packets of different lengths)

fluid-flow case where both queues have a one-unit length packet at time $t = 0$. The transmission of the packet from queue 2 is now completed at time $t = 4/3$, and the packet from queue 1 is completed at $t = 2$. The bit-by-bit approximation to weighted fair queueing would operate by allotting each queue a different number of bits/round. In the preceding example, queue 1 would receive 1 bit/round and queue 2 would receive 3 bits/round.

Weighted fair queueing is also easily approximated in ATM: in each round each nonempty queue would transmit a number of packets proportional to its weight. **Packet-by-packet weighted fair queueing** is also easily generalized from fair queueing. Suppose that there are n packet flows and that flow i has weight w_i, then the packet-by-packet system calculates its finish tag as follows:

$$F(i,k) = \max\{F(i,k-1), R(t_k^i)\} + P(i,k)/w_i \quad \text{for weighted fair queueing.}$$

Thus from the last term in the equation, we see that if flow i has a weight that is twice that of flow j, then the finish tag for a packet from flow i will be calculated assuming a depletion rate that is twice that of a packet from flow j.

Figure 7.49 also shows the completion times for the packet-by-packet weighted fair-queueing system. The finish tag of the packet from queue 1 is $F(1,1) = R(0) + 1/1 = 1$. The finish tag of the packet from queue 2 is $F(2,1) + R(0) + 1/3 = 1/3$. Therefore the packet from queue 2 is served first. The packet for queue 2 is now completed at time $t = 1$, and the packet from queue 1 at time $t = 2$. Note that packet-by-packet weighted fair queueing is also applicable when packets are of different length.

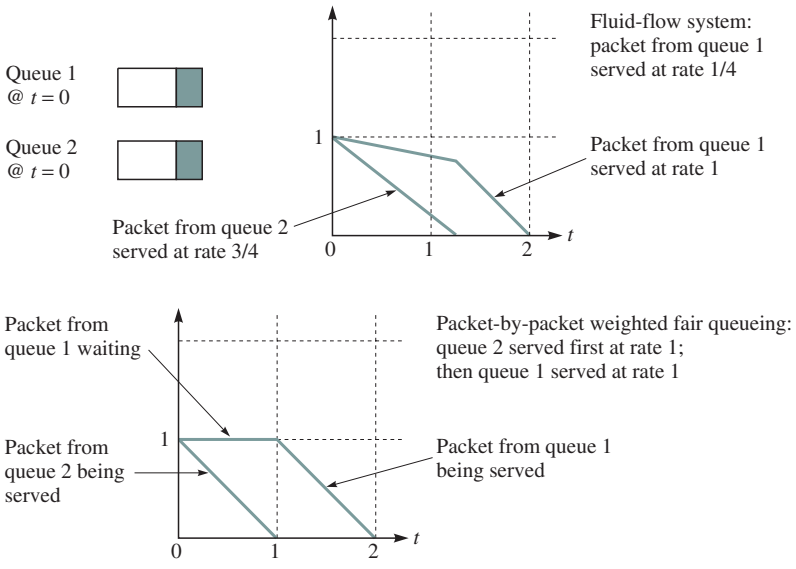


FIGURE 7.49 Fluid flow and packetized, weighted fair queueing

PROVIDING QoS IN THE INTERNET

In order to support real-time audio and video communications the Internet must provide some level of end-to-end QoS. One approach provides *differentiated service* in the sense that some classes of traffic are treated preferentially relative to other classes. This approach does not provide strict QoS guarantees. Packets are instead marked at the edge of the network to indicate the type of treatment that the packets are to receive in the routers inside the network. Modified forms of priority queueing can be used to provide the required differential treatment. A second approach provides *guaranteed service* that gives a strict bound on the end-to-end delay experienced by all packets that belong to a specific flow. This approach requires making resource reservations in the routers along the route followed by the given packet flow. Weighted fair queueing combined with traffic regulators are needed in the routers to provide this type of service. Differentiated service IP and guaranteed service IP are discussed in Chapter 10.

Weighted fair-queueing systems are a means for providing QoS guarantees. Suppose a given user flow has weight w_i and suppose that the sum of the weights of all the user flows is W. In the worst case when all the user queues are nonempty, the given user flow will receive a fraction w_i/W of the bandwidth C. When other user queues are empty, the given user flow will receive a greater share. Thus the user is guaranteed a minimum long-term bandwidth of at least $(w_i/W)C$ bps. This guaranteed share of the bandwidth to a large extent insulates the given user flow from the other user flows.

In addition, section 7.8 shows that if the user information arrival rate is *regulated* to satisfy certain conditions, then the maximum delay experienced in the multiplexer can be guaranteed to be below a certain value. In fact, it is possible to develop guaranteed bounds for the end-to-end delay across a series of multiplexers that use packet-by-packet weighted fair queueing. These bounds depend on the maximum burst that the user is allowed to submit at each multiplexer, on the weights at the various multiplexers, and on the maximum packet size that is allowed in the network. We return to the details of this scheme in section 7.8.

7.8 CONGESTION CONTROL

Congestion occurs when too many packets try to access the same buffer pool in a switch. For an example, consider the communication network shown in Figure 7.50. Suppose that nodes 1, 2, and 5 send bursts of packets to node 4 simultaneously. Assume that the aggregate incoming rate of the packets is greater than the rate at which the packets can be transmitted out. In this case the buffer in

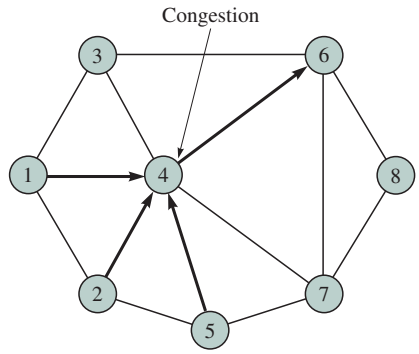


FIGURE 7.50 A congested switch

node 4 will build up. If this situation occurs sufficiently long, the buffer eventually may become full and start rejecting packets. When the destination detects the missing packets, it may ask the sources to retransmit the packets. The sources would unfortunately obey the protocol and send more packets to node 4, making the congestion even worse. In turn, node 4 discards more packets, and this effect triggers the destination to ask for more retransmissions. The net result is that the throughput at the destination will be very low, as illustrated in Figure 7.51 (uncontrolled curve). The purpose of congestion control is to eliminate or reduce congestion. If done properly, performance should improve (controlled curve).

For a novice, it is tempting to claim that congestion can be solved by just allocating a large buffer. However, this solution merely delays congestion from happening. Worse yet, when congestion kicks in, it will last much longer and will be more severe. In the worst case where the buffer size is infinite, packets can be delayed forever!

It turns out that congestion control is a very hard problem to solve. Typically, the solution depends on the application requirements (e.g., qualities of service). A variety of congestion control algorithms have been proposed in the literature. As with routing algorithms, we can classify congestion control algorithms several ways. The most logical approach is to divide them into two broad classes: *open loop* and *closed loop*. Open-loop algorithms prevent congestion from

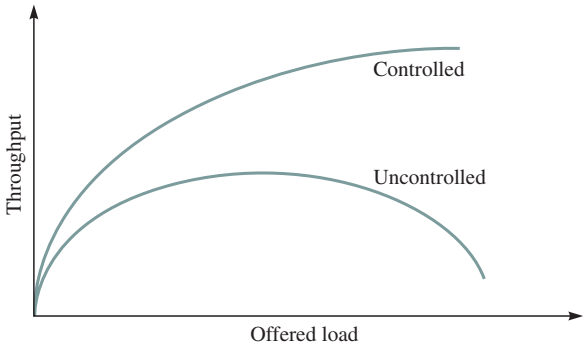


FIGURE 7.51 Throughput drops when congestion occurs

occurring by making sure that the traffic flow generated by the source will not degrade the performance of the network below the specified QoS. If the QoS cannot be guaranteed, the network has to reject the traffic flow. The function that makes the decision to accept or reject the traffic flow is usually called an *admission control*. Thus open-loop algorithms involve some type of resource reservation. Closed-loop algorithms, on the other hand, react to congestion when it is already happening or is about to happen, typically by regulating the traffic flow according to the state of the network. These algorithms are called closed loop because the state of the network has to be fed back to the point that regulates the traffic, which is usually the source. Closed-loop algorithms typically do not use any reservation.

It is important to note that congestion control algorithms are an effective way to reduce temporary overloads in the network (typically on the order of several milliseconds). If the overload lasts longer (several seconds to minutes), then adaptive routing may help by avoiding congested nodes and links. If the overload period is still longer, then the network has to be upgraded, for example, by deploying higher capacity links, faster switches, and so on.

7.8.1 Open-Loop Control

Open-loop congestion control does not rely on feedback information to regulate the traffic flow. Thus this technique assumes that once a source is accepted, its traffic flow will not overload the network. In this section we look at several promising open-loop approaches.

ADMISSION CONTROL

Admission control is an open-loop preventive congestion control scheme. It was initially proposed for virtual-circuit, packet-switched networks such as ATM but has been investigated for datagram networks as well. Admission control typically works at the connection level but can also work at the burst level. The analogy of a connection in datagram networks is a *flow*. At the connection level the function is called a connection admission control (CAC). At the burst level, it is called a burst admission control.

The main idea of CAC is very simple. When a source requests a connection setup, CAC has to decide whether to accept or reject the connection. If the QoS of all the sources (including the new one) that share the same path can be satisfied, the connection is accepted; otherwise, the connection is rejected. The QoS can be expressed in terms of maximum delay, loss probability, delay variance, and other performance parameters.

For CAC to determine whether the QoS can be satisfied, CAC has to know the traffic flow of each source. Thus each source must specify its traffic flow, described by a set of traffic parameters called the *traffic descriptor*, during the connection setup. A traffic descriptor may contain peak rate, average rate, maximum burst size, and so on, and is supposed to summarize the traffic flow compactly and accurately. Figure 7.52 shows an example of a traffic flow gen-

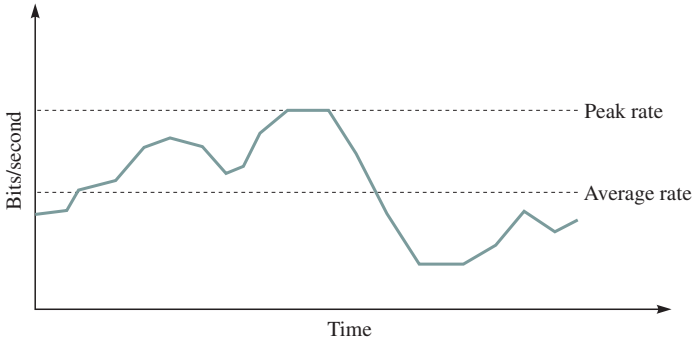


FIGURE 7.52 Example of a traffic flow

erated by a source, indicating the peak rate and the average rate. The maximum burst size usually relates to the maximum length of time the traffic is generated at the peak rate. Based on the characteristics of the traffic flow, CAC has to calculate how much bandwidth it has to reserve for the source. The amount of bandwidth typically lies between the average rate and the peak rate and is called the **effective bandwidth** of the source. The exact calculation for effective bandwidth is very complex and is beyond the scope of this book.

POLICING

Once a connection is accepted by a CAC, the QoS will be satisfied as long as the source obeys the traffic descriptor that it specified during the connection setup. However, if the traffic flow violates the initial contract, the network may not be able to maintain acceptable performance. To prevent the source from violating its contract, the network may want to monitor the traffic flow during the connection period. The process of monitoring and enforcing the traffic flow is called *traffic policing*. When the traffic violates the agreed-upon contract, the network may choose to discard or tag the nonconforming traffic. The tagged traffic will be carried by the network but given lower priority. If there is any congestion downstream, the tagged traffic is the first one to be lost.

Most implementations of traffic policing use the **leaky bucket** algorithm. To understand how a leaky bucket can be used as a policing device, imagine the traffic flows to a policing device as water being poured into a bucket that has a hole at the bottom, as illustrated in Figure 7.53. The bucket has a certain depth and leaks at a constant rate when it is not empty. A new container (that is, packet) of water is said to be *conforming* if the bucket does not overflow when the water is poured in the bucket. The bucket will spill over if the amount of water in the container is too large or if the bucket is nearly full from prior containers. The bucket depth is used to absorb the irregularities in the water flow. If we expect the traffic flow to be very smooth, then the bucket can be made very shallow. If the flow is bursty, the bucket should be deeper. The drain rate corresponds to the traffic rate that we want to police.

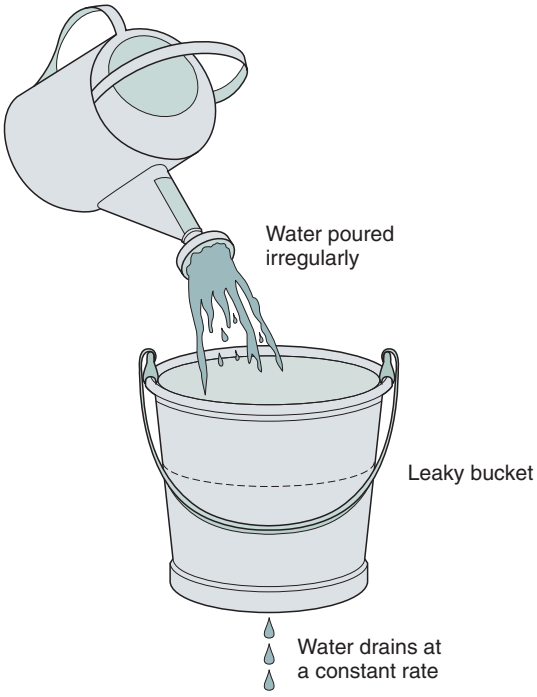


FIGURE 7.53 A leaky bucket

There are many variations of the leaky bucket algorithm. In this section we look at an algorithm that is standardized by the ATM Forum. Here packets are assumed to be of fixed length (i.e., ATM cells). A counter records the content of the leaky bucket. When a packet arrives, the value of the counter is incremented by some value I provided that the content of the bucket would not exceed a certain limit; in this case the packet is declared to be conforming. If the content would exceed the limit, the counter remains unchanged and the packet is declared to be nonconforming. The value I typically indicates the nominal interarrival time of the packet that is being policed (typically, in units of packet time). As long as the bucket is not empty, the bucket will drain at a continuous rate of 1 unit per packet time.

Figure 7.54 shows the leaky bucket algorithm that can be used to police the traffic flow. At the arrival of the first packet, the content of the bucket X is set to zero and the last conforming time (LCT) is set to the arrival time of the first packet. The depth of the bucket is $L + I$, where L depends on the maximum burst size. If the traffic is expected to be bursty, then the value of L should be made large. At the arrival of the kth packet, the auxiliary variable X' records the difference between the bucket content at the arrival of the last conforming packet and the interarrival time between the last conforming packet and the kth packet. The auxiliary variable is constrained to be nonnegative. If the auxiliary variable is greater than L, the packet is considered nonconforming. Otherwise, the packet is conforming. The bucket content and the arrival time of the packet are then updated.

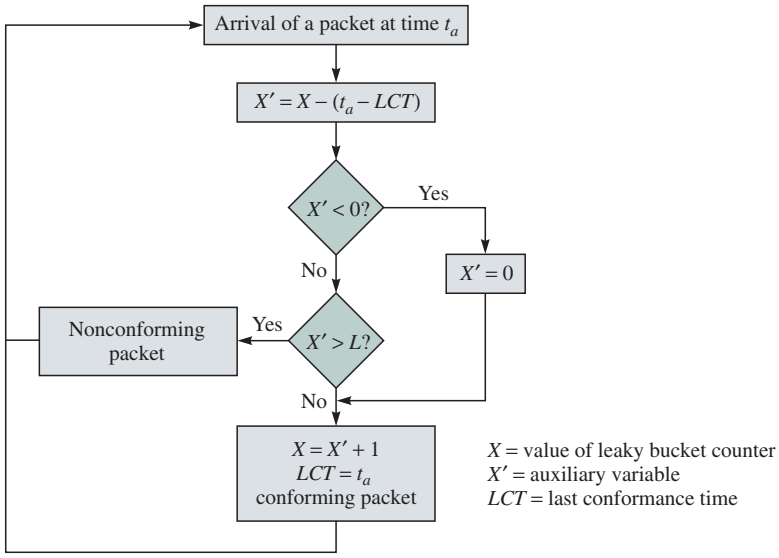


FIGURE 7.54 Leaky bucket algorithm used for policing

A simple example of the operation of the leaky bucket algorithm is shown in Figure 7.55. Here the value of I is four packet times, and the value of L is six packet times. The arrival of the first packet increases the bucket content by four (packet times). At the second arrival the content has decreased to three, but four more are added to the bucket resulting in a total of seven. The fifth packet is declared as nonconforming since it would increase the content to 11, which would exceed $L + I$. Packets 7, 8, 9, and 10 arrive back to back after the bucket becomes empty. Packets 7, 8, and 9 are conforming, and the last one is nonconforming. If the peak rate is one packet/packet time, then the **maximum burst size (MBS)** for this algorithm is three. Note that the algorithm does not update the content of the bucket continuously, but only at discrete points (arrival times) indicated by the asterisks. Also note that the values of I and L in general can take any real numbers.

Often the inverse of I is the *sustainable rate* which is the long-term average rate allowed for the conforming traffic. Suppose the *peak rate* of a given traffic is denoted by R and its inverse is T; that is, $T = 1/R$. Then the maximum burst size is given by

$$MBS = 1 + \left[\frac{L}{1 - T}\right] \tag{10}$$

where $[x]$ gives the greatest integer less than or equal to x. To understand this formula, note that the first packet increases the bucket content to I. After the first packet the bucket content increases by the amount of $(I - T)$ for each packet arrival at the peak rate. Thus we can have approximately $L/(I - T)$

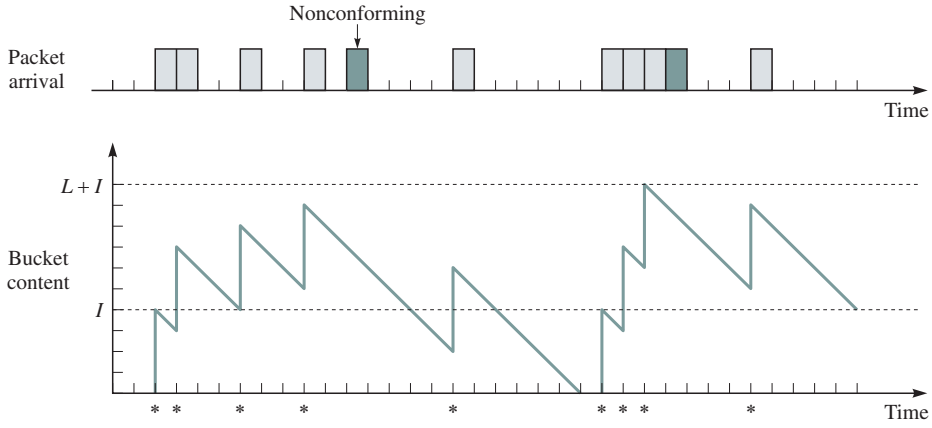


FIGURE 7.55 Behavior of leaky bucket

additional conforming packets. The relations among these quantities are pictorially depicted in Figure 7.56. MBS roughly characterizes the burstiness of the traffic. Bursty traffic may be transmitted at the peak rate for some time and then remains dormant for a relatively long period before being transmitted at the peak rate again. This type of traffic tends to stress the network.

Leaky buckets are typically used to police both the peak rate and the sustainable rate. In this situation *dual leaky buckets* such as the one shown in Figure 7.57 can be used. The traffic is first checked for the peak rate at the first leaky bucket. The cell delay variation tolerance (CDVT) is the amount of variation that is allowed in the peak cell rate. The bucket has a total capacity of T and τ and each arrival of a conforming packet increases the bucket by T. The nonconforming packets at the first bucket are dropped or tagged. The conforming (untagged) packets are then checked for the sustainable rate at the second leaky bucket. The nonconforming packets at the second leaky bucket are also dropped or tagged. The conforming packets are the ones that remain untagged after both leaky buckets.

TRAFFIC SHAPING

When a source tries to send packets, it may not know what its traffic looks like. If the source wants to ensure that the traffic conforms to the parameters specified in the leaky bucket policing device, it should first alter the traffic. The process of altering a traffic flow to another flow is called **traffic shaping**.

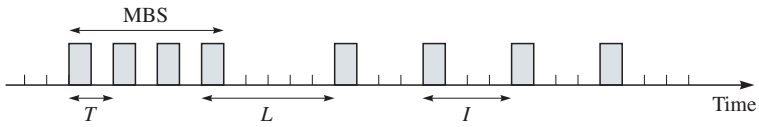


FIGURE 7.56 Relations among MBS and other parameters

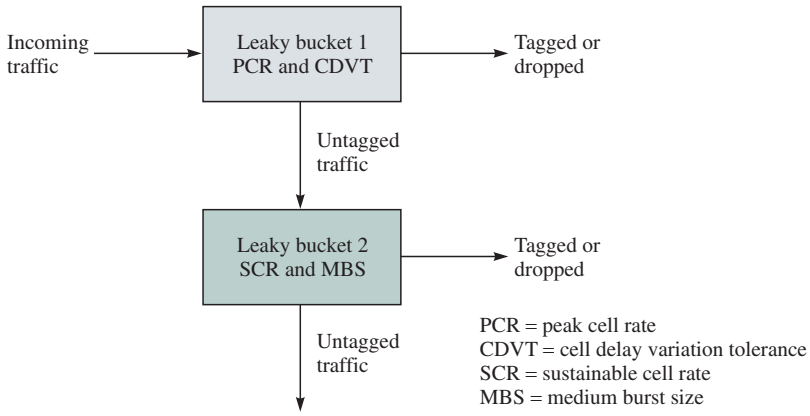


FIGURE 7.57 A dual leaky bucket configuration

Traffic shaping can also be used to make the traffic smoother. Consider an example where an application periodically generates 10 kilobits of data every second. The source can transmit the data in many ways. For example, it can transmit at the rate of 10 kbps continuously. It can transmit at the rate of 50 kbps for 0.2 seconds for each period or at the rate of 100 kbps for 0.1 second for each period, as illustrated in Figure 7.58. From the network's point of view, the traffic shown in Figure 7.58a represents the smoothest pattern, and the one least likely to stress the network. However, the destination may not want to wait for 1 second to receive the data at each period. Another use of traffic shaping is to smooth the traffic flow according to the user's specification.

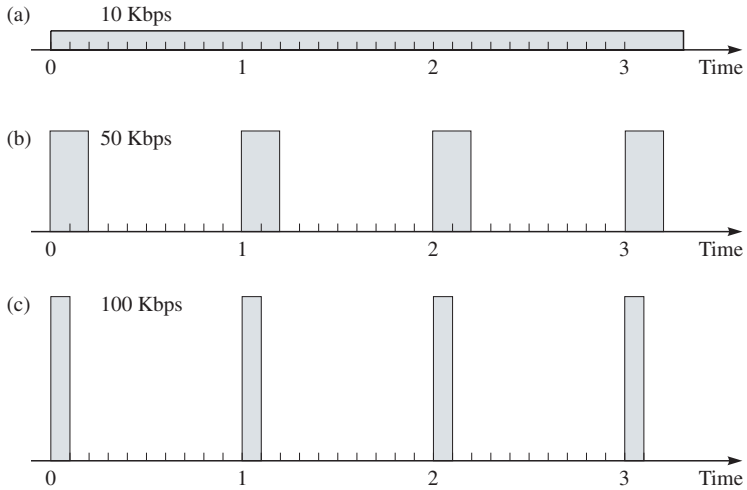


FIGURE 7.58 Possible traffic patterns at the average rate of 10 kbps

There are many implementations of traffic shaping. In this section, we will look at two possibilities. The first one is based on a leaky bucket. The second is usually called a token bucket shaper.

Leaky Bucket Traffic Shaper

A *leaky bucket traffic shaper* is a very simple device. It can be implemented by a buffer whose content is read out periodically at a constant interval, as shown in Figure 7.59. Unlike the leaky bucket policing algorithm, which only monitors the traffic, a leaky bucket traffic shaper regulates the traffic flow. The bucket in the policing algorithm is just a counter, whereas the bucket in the shaper is a buffer that stores packets.

Incoming packets are first stored in a buffer. Packets are served periodically so that the stream of packets at the output is smooth. The buffer is used to store momentary bursts of packets. The buffer size defines the maximum burst that can be accommodated. If the buffer is full, incoming packets are in violation and are thus discarded.

Token Bucket Traffic Shaper

The leaky bucket traffic shaper described above is very restricted, since the output rate is constant when the buffer is not empty. Many applications produce variable-rate traffic. If such applications have to go through the leaky bucket traffic shaper, the delay through the buffer can be unnecessarily long. Recall that the traffic that is monitored by the policing algorithm does not have to be smooth to be conforming. The policing device allows for some burstiness in the traffic as long as it is under a certain limit.

Another more flexible shaper, called the *token bucket traffic shaper*, regulates only the packets that are not conforming. Packets that are deemed conforming are passed through without further delay. In Figure 7.60 we see that the token bucket is a simple extension of the leaky bucket. Tokens are generated periodically at a constant rate and are stored in a token bucket. If the token bucket is full, arriving tokens are discarded. A packet from the buffer can be taken out only if a token in the token bucket can be drawn. If the token bucket is empty, arriving packets have to wait in the packet buffer. Thus we can think of a token as a permit to send a packet.

Imagine that the buffer has a backlog of packets when the token bucket is empty. These backlogged packets have to wait for new tokens to be generated

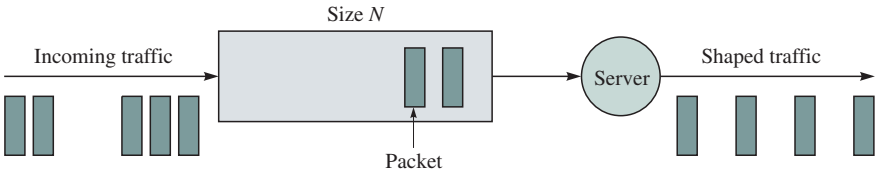


FIGURE 7.59 A leaky bucket traffic shaper

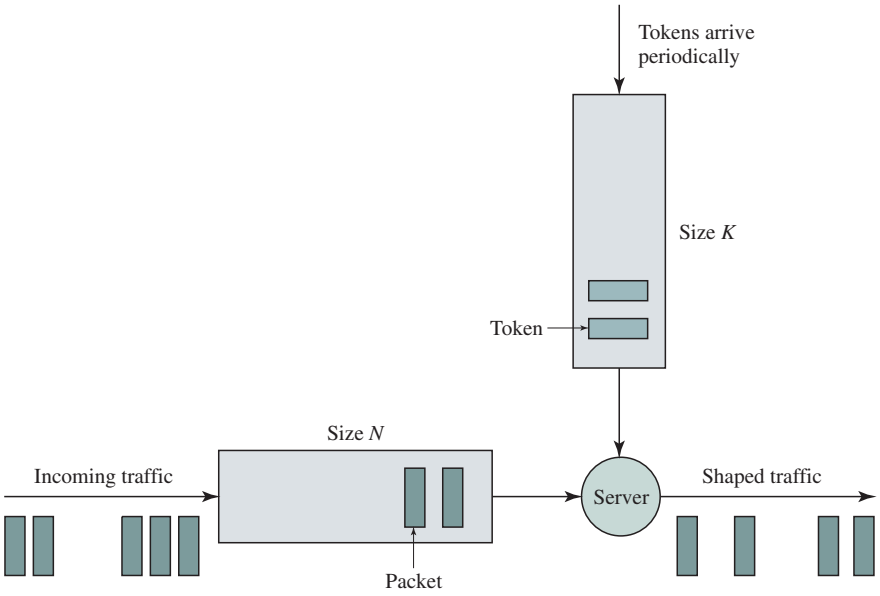


FIGURE 7.60 Token bucket traffic shaper

before they can be transmitted out. Since tokens arrive periodically, these packets will be transmitted periodically at the rate the tokens arrive. Here the behavior of the token bucket shaper is very similar to that of the leaky bucket shaper.

Now consider the case when the token bucket is not empty. Packets are transmitted out as soon as they arrive without having to wait in the buffer, since there is a token to draw for an arriving packet. Thus the burstiness of the traffic is preserved in this case. However, if packets continue to arrive, eventually the token bucket will become empty and packets will start to leave periodically. The size of the token bucket essentially limits the traffic burstiness at the output. In the limit, as the bucket size is reduced to zero, the token bucket shaper becomes a leaky bucket shaper.

QoS GUARANTEES AND SERVICE SCHEDULING

Switches and routers in packet-switched networks use buffers to absorb temporary fluctuations of traffic. Packets that are waiting in the buffer can be scheduled to be transmitted out in a variety of ways. In this section we discuss how the packet delay across a network can be guaranteed to be less than a given value. The technique makes use of a token bucket shaper and weighted fair-queueing scheduling.

Let b be the bucket size in bytes and let r be the token rate in bytes/second. Then in a time period T, the maximum traffic that can exit the traffic shaper is $b + rT$ bytes as shown in Figure 7.61. Suppose we apply this traffic to two multiplexers in tandem each served by transmission lines of speed R bytes/second

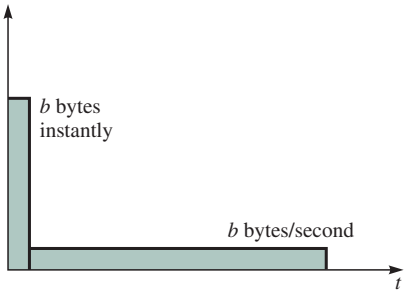


FIGURE 7.61 Maximum traffic allowed out of token bucket shaper

with $R > r$. We assume that the two multiplexers are empty and not serving any other flows.

Figure 7.62a shows the multiplexer arrangement, and Figure 7.62b shows the buffer contents as a function of time. We assume that the token bucket allows an immediate burst of b bytes to exit and appear at the first multiplexer at $t = 0$, so the multiplexer buffer surges to b bytes at that instant. Immediately after $t = 0$, the token bucket allows information to flow to the multiplexer at a rate of r bytes/second, and the transmission line drains the multiplexer at a rate of R bytes/second. Thus the buffer occupancy falls at a rate of $R - r$ bytes/second. Note that the buffer occupancy at a given instant determines the delay that will be experienced by a byte that arrives at that instant, since the occupancy is exactly the number of bytes that need to be transmitted before the arriving byte is itself transmitted. Therefore, we conclude that the maximum delay at the first multiplexer is bounded by b/R.

Now consider the second multiplexer. At time $t = 0$, it begins receiving bytes from the first multiplexer at a rate of R bytes/second. The second multiplexer immediately begins transmitting the arriving bytes also at a rate of R bytes/second. Therefore there is no queue buildup in the second multiplexer, and the byte stream flows with zero queueing delay. We therefore conclude that the

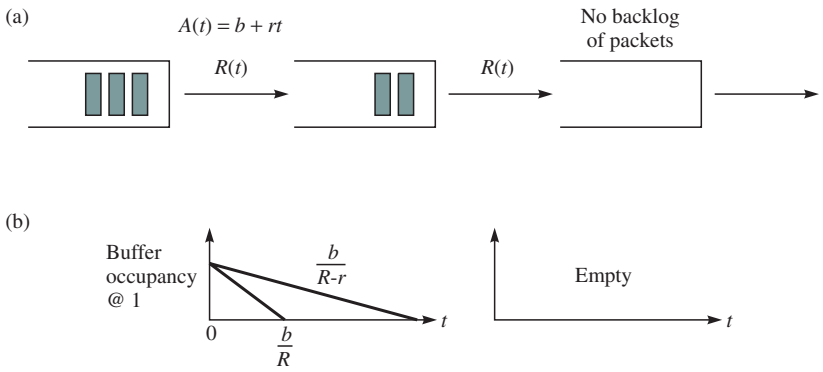


FIGURE 7.62 Delay experienced by token bucket shaped traffic

information that exits the token bucket shaper will experience a delay no greater than b/R over the chain of multiplexers.

Suppose that the output of the token bucket shaper is applied to a multiplexer that uses weighted fair queueing. Also suppose that the weight for the flow has been set so that it is guaranteed to receive at least R bytes/second. It then follows that the flow from the token bucket shaper will experience a delay of at most b/R seconds. This result, however, assumes that the byte stream is handled as a fluid flow. [Parekh 1992] showed that if packet-by-packet weighted fair queueing is used, then the maximum delay experienced by packets that are shaped by a (b, r) token bucket and that traverse H hops is bounded as follows:

$$D \leq \frac{b}{R} + \frac{(H-1)m}{R} + \sum_{j=1}^{H} \frac{M}{R_j} \tag{11}$$

where m is the maximum packet size for the given flow, M is the maximum packet size in the network, H is the number of hops, and R_j is the speed of the transmission line in link j. Also note that $r \leq R$. This result provides the basis for setting up connections across a packet network that can guarantee the packet delivery time. This result forms the basis for the *guaranteed delay service* proposal for IP networks.

To establish a connection that can meet a certain delay guarantee, the call setup procedure must identify a route in which the links can provide the necessary guaranteed bandwidth so that the bound is met. This process will involve obtaining information from potential hops about their available bandwidth, selecting a path, and allocating the appropriate bandwidth in the path.

7.8.2 Closed-Loop Control

Closed-loop congestion control relies on feedback information to regulate the source rate. The feedback information can be implicit or explicit. In the implicit feedback the source may use a time-out to decide whether congestion has occurred in the network. In the explicit feedback some form of explicit message will arrive at the source to indicate the congestion state in the network. In the next two subsections, we discuss the closed-loop control used in TCP and in ATM networks. It is interesting to note that TCP exercises congestion control at the transport layer, whereas ATM operates at the network layer.

TCP Congestion Control

Recall from Chapter 5 that TCP uses a sliding-window protocol for end-to-end flow control. This protocol is implemented by having the receiver specify in its acknowledgment the amount of bytes it is willing to receive in the future, called the *advertised window*. The advertised window ensures that the receiver's buffer will never overflow, since the sender cannot transmit data that exceeds the

amount that is specified in the advertised window. However, the advertised window does not prevent the buffers in the intermediate routers from overflowing—the condition is called congestion. Routers can become overloaded when they have to cope with too many packets in their buffers. Because IP does not provide any mechanism to control congestion, it is up to the higher layer to detect congestion and take proper actions. It turns out that TCP window mechanism can also be used to control congestion in the network.

The basic idea of TCP congestion control is to have each sender transmit just the right amount of data to keep the network resources utilized but not overloaded. If the senders are too aggressive by sending too many packets, the network will experience congestion. On the other hand, if TCP senders are too conservative, the network will be underutilized. The maximum amount of bytes that a TCP sender can transmit without congesting the network is specified by another window called the *congestion window*. To avoid network congestion and receiver buffer overflow, the maximum amount of data that a TCP sender can transmit at any time is the minimum of the advertised window and the congestion window.

The TCP congestion control algorithm dynamically adjusts the congestion window according to the network state. The operation of the TCP congestion control algorithm may be divided into three phases. The first phase is run when the algorithm starts or restarts, assuming that the pipe is empty. The technique is called *slow start* and is accomplished by first setting the congestion window to one maximum-size segment.[9] Each time the sender receives an acknowledgment from the receiver, the sender increases the congestion window by one segment. After sending the first segment, if the sender receives an acknowledgment before a time-out, the sender increases the congestion window to two segments. If these two segments are acknowledged, the congestion window increases to four segments, and so on. As shown in Figure 7.63, the congestion window size grows exponentially during this phase. The reason for the exponential increase is that slow start needs to fill an empty pipe as quickly as possible. The name ''slow start'' is perhaps a misnomer, since the algorithm ramps up very quickly.

Slow start does not increase the congestion window exponentially forever, since the pipe will be filled up eventually. Specifically, slow start stops when the congestion window reaches a value specified as the *congestion threshold* which is initially set to 65,535 bytes. At this point a *congestion avoidance* phase takes over. This phase assumes that the pipe is running close to full utilization. It is wise for the algorithm to reduce the rate of increase so that it will not overshoot excessively. Specifically, the algorithm increases the congestion window linearly rather than exponentially during congestion avoidance. This is realized by increasing the congestion window by one segment for each round-trip time.

Obviously, the congestion window cannot be increased indefinitely. The congestion window stops increasing when TCP detects that the network is congested. The algorithm now enters the third phase. At this point the congestion

[9] Recall from Chapters 2 and 5 that a segment is the data block or protocol data unit that is used by TCP.

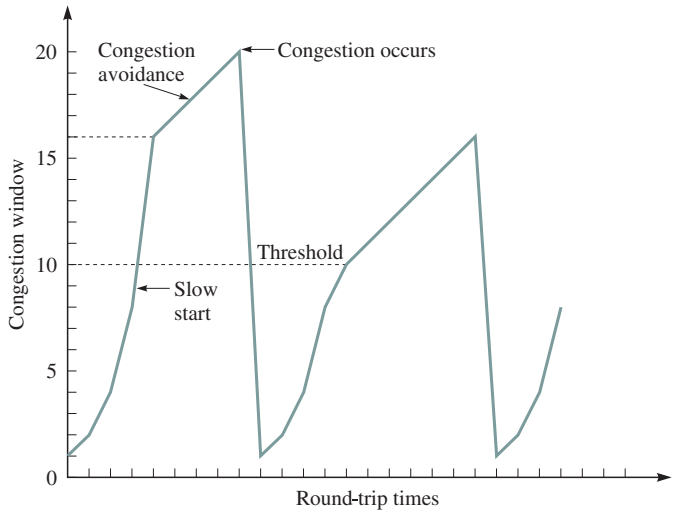


FIGURE 7.63 Dynamics of TCP congestion window

threshold is first set to one-half of the current window size (the minimum of the congestion window and the advertised window, but at least two segments). Next the congestion window is set to one maximum-sized segment. Then the algorithm restarts, using the slow start technique.

How does TCP detect that the network is congested? TCP assumes that congestion occurs in a network when an acknowledgment does not arrive before the time-out expires because of segment loss. The basic assumption the algorithm is making is that a segment loss is due to congestion rather than errors. This assumption is quite valid in a wired network where the percentage of segment losses due to transmission errors is generally low (less than 1 percent). However, it should be noted that the assumption may not be valid in a wireless network where transmission errors can be relatively high.

TCP also detects congestion when a duplicate ACK is received, which can be due to either segment reordering or segment loss. TCP reacts to duplicate ACKs by decreasing the congestion threshold to one-half the current window size, as before. However, the congestion window is *not* reset to one. If the congestion window is less than the new congestion threshold, then the congestion window is increased as in slow start. Otherwise, the congestion window is increased as in congestion avoidance.[10]

Figure 7.63 illustrates the dynamics of the congestion window as time progresses. Initially, slow start kicks in until the congestion threshold (set at 16 segments) is reached. Then congestion avoidance increases the window linearly until a time-out occurs, indicating that the network is congested. The congestion

[10]See RFC 2001 for details of the operation.

threshold is set to 10 segments, and the congestion window is set to 1 segment. The algorithm then starts with slow start again.

Our discussion so far assumed that the TCP entity knows when to time out. It turns out that computing the retransmission time-out value (RTO) is not a trivial problem, since network delays are highly variable. In general, we can roughly say that the queueing delay in a router is proportional to $1/(1-\rho)$, where ρ is the traffic load. Thus the round-trip time (RTT) that is used as the basis for computing RTO varies over time, depending on the traffic load. Each time TCP receives an acknowledgment to a segment, TCP records the current measured RTT and then updates the estimate of RTT. There are two methods for estimating RTT for TCP. The older method involved estimating the time-out RTO as a fixed multiple of RTT; that is, RTO = β RTT. Early implementations of TCP used a constant value of β equal to 2. However, the constant value was found to be inadequate as it does not take into account the delay variance. Typically, the delay variance in a queue is proportional to $1/(1-\rho)^2$. This expression says that when the network is lightly loaded, RTT is almost constant (small variance). When the network is heavily loaded, RTT varies widely (large variance).

In the late 1980s Jacobson proposed an efficient implementation that keeps track of the delay variance [Jacobson 1988]. Specifically, the algorithm estimates a *standard deviation* using a *mean deviation*. The mean deviation gives a somewhat more conservative result than the standard deviation but is much easier to compute. The smoothed mean deviation, DEV, is computed according to

$$DEV \leftarrow \delta DEV + (1-\delta)|RTT - M| \tag{12}$$

where δ is typically set to 3/4. Once the mean deviation of the round-trip time is found, the RTO is finally set to

$$RTO \leftarrow RTT + 4DEV \tag{13}$$

An ambiguity exists when a segment is retransmitted because the time-out has expired. When the acknowledgment eventually comes back, do we associate it with the first segment or with the retransmitted segment? Karn proposed ignoring the RTT when a segment is retransmitted because using the RTT may corrupt the estimate. This idea is generally called *Karn's algorithm*.

ABR CONGESTION CONTROL FOR ATM NETWORKS

The **available bit rate (ABR)** service in ATM is intended for non-real-time applications such as data communications. ABR service does not have a strict delay or loss constraint. However, the network would try to minimize the cell loss ratio, since a typical data payload would contain many ATM cells and a loss of an ATM cell ruins the entire payload. At connection setup an ABR source is required to specify its peak cell rate (PCR) and its minimum cell rate (MCR), which may be set to zero. During data transfer the network tries to give up as much bandwidth as possible to the PCR but never goes below the MCR. The bandwidth available to ABR sources at any time depends on the residual band-

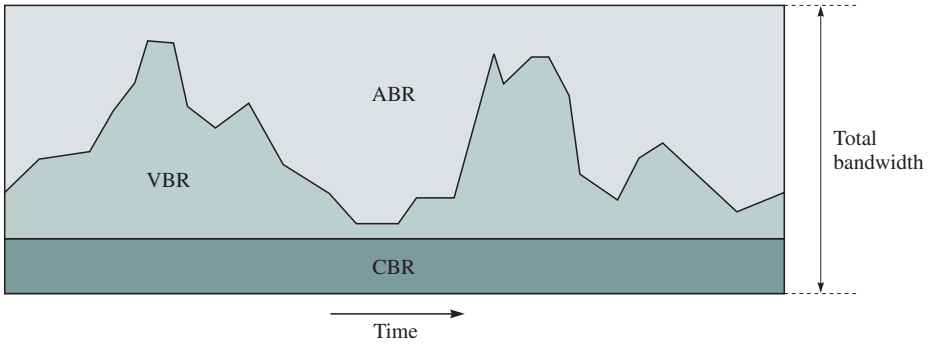


FIGURE 7.64 Bandwidth allocation to services

width in the pipe after the bandwidth for constant bit rate (CBR) and variable bit rate (VBR) sources have been allocated, as shown in Figure 7.64.

ABR congestion control in ATM network uses a rate-based control mechanism that works by continuously adjusting the source rate according to the network state. The information about the network state is carried by special control cells called **resource management (RM)** cells. An ABR source generates forward RM cells periodically at the rate of one RM cell for every $N_{RM} - 1$ of data cells, as shown in Figure 7.65. At the destination the RM cells are turned around and sent back to the source. The backward RM cells carry the feedback information used by the source to control its rate. The way the feedback information is inserted in the RM cell depends on implementations. The next two sections describe possible implementations.

Binary Feedback

The binary feedback scheme allows switches with minimal functionalities to participate in the control loop. Each source sends its data cells with the explicit forward congestion indication (EFCI) bit set to zero to indicate that no congestion is experienced. RM cells that are sent periodically also have the

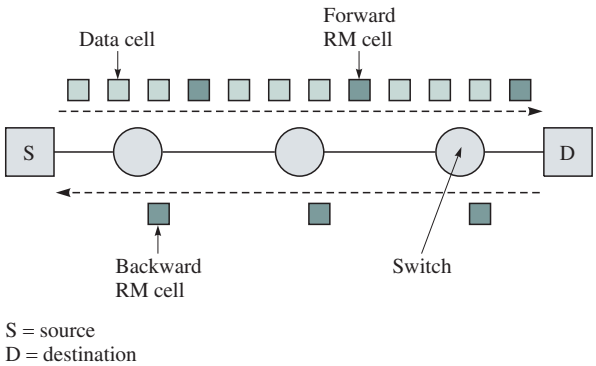


FIGURE 7.65 End-to-end rate-based control

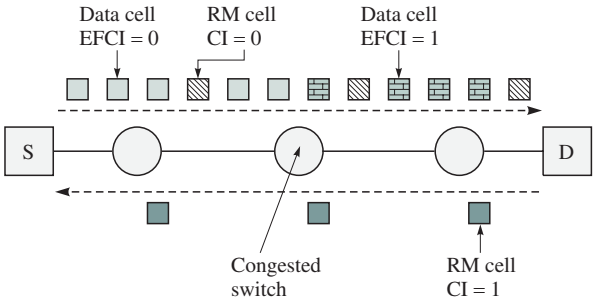


FIGURE 7.66 Binary feedback control

congestion indication (CI) bit set to zero. Each switch along the connection monitors the link congestion status continuously. A switch may decide that the link is congested if the associated queue level exceeds a certain threshold. In this case the switch would set the EFCI bit of all data cells passing through the queue to one, as shown in Figure 7.66.

When data cells received by the destination have the EFCI bit set to one, the destination should set the CI bit of the backward RM cell to one, indicating that the forward path is congested. Else, the CI bit is left unchanged. Note that the resulting technique conveys binary feedback information only, since it can only tell the source whether or not there is congestion in the path.

How does the source react to congestion? When the source receives a backward RM cell with the CI bit set to zero, that source could increase its transmission rate. If the CI bit is one, the source should instead decrease its transmission rate. Typically, the increase would be linear, whereas the decrease would be exponential. In the absence of backward RM cells, the source should decrease its transmission rate. This method, sometimes called *positive feedback*, essentially forces the source to decrease its rate in case the backward path is congested and thus provides a more robust control.

Explicit Rate Feedback

Because the binary feedback scheme can only tell the source to increase or decrease its rate, the scheme tends to converge slowly and oscillate widely around the operating point. Another method, called the *explicit rate* scheme, tries to solve the convergence problem by allowing each switch to explicitly indicate the desired rate to the RM cell that passes through.

The explicit rate scheme works as follows. Each source puts the rate at which it would like to transmit cells in the explicit rate (ER) field of the forward RM cell. The value of the ER field is initially set to PCR. Any switch along the path may reduce the ER value to the desired rate that it can support. However, a switch must not increase the ER value, since doing so would nullify the value set by the more congested switch. If the destination is congested, it may also reduce the ER value before returning the RM cell to the source. When the source receives the backward RM cell, the source adjusts its transmission rate so as not to exceed the ER value. By doing so, every switch along the path is expected

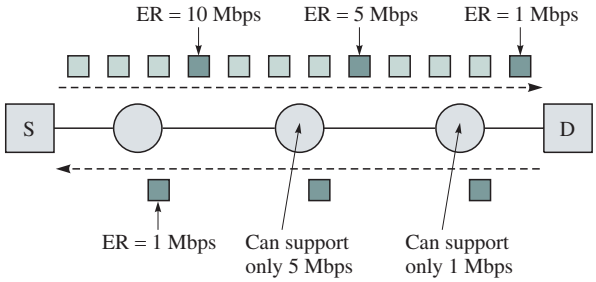


FIGURE 7.67 Explicit rate control

to be able to support the rate. Figure 7.67 illustrates the operation of the ER control.

The switch can use many methods to compute its desired rate. One attractive way, which is based on the *max-min fairness* principle, assigns the available bandwidth equally among connections that are bottlenecked on the considered link. If B is the bandwidth to be shared by active connections bottlenecked on the link and n is the number of active connections bottlenecked on the link, then the fair share for connection i, $B(i)$, is

$$B(i) = \frac{B}{n} \tag{14}$$

Enhanced Proportional Rate Control Algorithm

The **enhanced proportional rate control algorithm (EPRCA)** combines both binary feedback and ER feedback schemes, allowing simple switches supporting only EFCI bit setting to interoperate with more complex switches that can compute ERs.

Switches that implement only the EFCI mechanism would ignore the content of the RM cell and would set the EFCI bit to one if the link is congested. Switches that implement the ER scheme may reduce the ER value in the RM cell accordingly if the link is congested. The destinations turn around RM cells, setting the CI bit to one if the last received data cell has the EFCI bit set to one. When the source receives the backward RM cell, the source would set its transmission rate to the minimum value calculated by the binary feedback scheme and the ER value specified in the RM cell.

In EPRCA an ABR source should adhere to the following rules:

1. The source may transmit cells at any rate up to the *allowed cell rate* (ACR). The value of the ACR should be bounded between MCR and PCR.
2. At the call setup time, the source sets ACR to the *initial cell rate* (ICR). The first cell transmitted is an RM cell. When the source has been idle for some time, ACR should also be reduced to ICR.
3. The source should send one RM cell for every $N_{\mathrm{RM}} - 1$ data cells or when T_{RM} time (typically set to 100 msec) has elapsed.

4. If the backward RM cell does not return, the source should decrease its ACR by ACR * RDF, down to MCR. RDF stands for rate decrease factor and is typically set to 1/16.
5. When the source receives a backward RM cell with CI = 1, the source should also decrease its ACR by ACR * RDF, down to MCR.
6. When the source receives a backward RM cell with CI = 0, the source may increase the ACR by no more than RIF * PCR, up to the PCR. RIF stands for rate increase factor and is typically 1/16.
7. When the source receives any backward RM cell, the source should set the ACR to the minimum of the ER value from the RM cell and the ACR computed in 5 and 6.

An ABR destination should adhere to the following rules:

1. The destination should turn around all RM cells so that they can return to the source. The direction bit (DIR) in the RM cell should be set to one to indicate a backward RM cell.
2. If the last received data cell prior to a forward RM cell had an EFCI bit set to one, the destination should set the CI bit in the backward RM cell to one. The destination may also reduce the ER value to whatever rate it can support.

Finally, an ATM switch supporting ABR congestion control should adhere to the following rules:

1. The switch should implement either EFCI marking or ER marking. With EFCI marking the switch should set the EFCI bit of a data cell to one when the link is congested. With ER marking the switch may reduce the ER field of forward or backward RM cells.
2. The switch may set the CI bit of the backward RM cell to one to prevent the source from increasing its rate.
3. The switch may generate a backward RM cell to make the source respond faster. In such a case, the switch should set CI = 1 and BN = 1 to indicate that the RM cell is not generated by the source.

SUMMARY

In this chapter we have examined networks that transfer information in the form of blocks called packets. We began with a discussion of how packet information is transferred end-to-end across the Internet. We saw how the Internet protocol achieves this transfer over a variety of networks including LANs and ATMs. We also how the Internet involves the distributed operation of multiple autonomous domains.

Packet networks can offer either connection-oriented service or connectionless services to the transport layer. These services are supported by the internal operation of the packet network, which can involve virtual circuits and data-

grams. The Internet is an example of a network that operates internally using datagrams. ATM networks provide an example of virtual-circuit operation. We discussed the advantages and disadvantages of virtual circuits and datagrams in terms of their complexity, their flexibility in dealing with failures, and their ability to provide QoS. We also discussed the structure of generic switches and routers.

We next considered the key topic of routing. We explained the use of routing tables in the selection of the paths that packets traverse in the virtual circuit and in datagram networks. We introduced approaches to synthesizing routing tables and we developed corresponding shortest path algorithms. We also discussed the interplay between hierarchical addressing and routing table size.

We then returned the topic of ATM networks and explained in more detail the use of VCIs in establishing end-to-end connections across a network. We also explained the use of VPIs in managing bandwidth and in creating logically-defined network topologies.

FIFO, priority, and weighted fair queueing mechanisms were introduced and their role in traffic management to provide QoS in packet networks was explained. The combination of traffic regulators and weighted fair queueing to provide guaranteed delay service was also discussed.

Finally, control techniques for dealing with congestion in the network were introduced. We showed how TCP provides for congestion control in the Internet using end-system mechanisms. We also showed how rate-based mechanisms provide congestion control in ATM networks.

CHECKLIST OF IMPORTANT TERMS

asynchronous transfer mode (ATM)
available bit rate (ABR)
Bellman-Ford algorithm
cell
centralized routing
connectionless
connection-oriented
counting to infinity
cut-through packet switching
datagram packet switching
deflection routing
Dijkstra's algorithm
distributed routing
dynamic (adaptive) routing
effective bandwidth
enhanced proportional rate control algorithm (EPRCA)
fair queueing
finish tag
first-in, first-out (FIFO) queueing
flooding
head-of-line (HOL) priority queueing
leaky bucket
load
Manhattan street network
maximum burst size (MBS)
message switching
network routing
packet switching
packet-by-packet fair queueing
packet-by-packet weighted fair queueing
packets
resource management (RM)

router
shortest-path algorithms
source routing
split horizon
split horizon with poisoned reverse
static routing
traffic shaping
virtual channel identifier (VCI)
virtual-circuit connection (VCC)
virtual-circuit identifier (VCI)
virtual-circuit packet switching
virtual path
virtual path identifier (VPI)
weighted fair queueing

FURTHER READING

Bertsekas, D. and R. Gallagher, *Data Networks*, Prentice-Hall, Englewood Cliffs, New Jersey, 1992.

Halabi, B., *Internet Routing Architectures*, New Riders Publishing, Indianapolis, Indiana, 1997.

Hashemi, M. R. and A. Leon-Garcia, "Implementation of Scheduling Schemes Using a Sequence Circuit," *Voice, Video, and Data Communications*, SPIE, Dallas, November 1997.

Huitema, C., *Routing in the Internet*, Prentice-Hall, Englewood Cliffs, New Jersey, 1995.

Jacobson, V., "Congestion Avoidance and Control," *Computer Communications Review*, Vol. 18, No. 4, August 1988, pp. 314–329.

Keshav, S., *An Engineering Approach to Computer Networking*, Addison-Wesley, Reading, Massachusetts, 1997.

McDysan, D. E. and D. L. Spohn, *ATM: Theory and Application*, McGraw-Hill, New York, 1995.

Parekh, A. K., "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks," Ph.D. dissertation, Department of Electrical Engineering and Computer Science, MIT, February 1992.

Perlman, R., *Interconnections: Bridges and Routers*, Addison-Wesley, Reading, Massachusetts, 1992. (The most comprehensive book on bridges and routers.)

Robertazzi, T. G., *Performance Evaluation of High Speed Switching Fabrics*, IEEE Press, 1994.

Saltzer, J. et al., "End-to-end arguments in System Design," *ACM Transactions on Computer Systems*, Vol. 2, No. 4, November 1984, pp. 277–288.

Zhang, H., "Service Disciplines for Guaranteed Performance in Packet Switching Networks," *Proceedings of IEEE*, October 1995, pp. 1374–1396.

RFC 2001, W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," January 1997.

PROBLEMS

1. Explain how a network that operates internally with virtual circuits can provide connectionless service. Comment on the delay performance of the service. Can you identify inefficiencies in this approach?

2. Is it possible for a network to offer best-effort virtual-circuit service? What features would such a service have, and how does it compare to best-effort datagram service?

3. Suppose a service provider uses connectionless operation to run its network internally. Explain how the provider can offer customers reliable connection-oriented network service.

4. Where is complexity concentrated in a connection-oriented network? Where is it concentrated in a connectionless network?

5. Comment on the following argument: Because they are so numerous, end systems should be simple and dirt cheap. Complexity should reside inside the network.

6. In this problem you compare your telephone demand behavior and your Web demand behavior.
   a. Arrival rate: Estimate the number of calls you make in the busiest hour of the day; express this quantity in calls/minute. Service time: Estimate the average duration of a call in minutes. Find the load that is given by the product of arrival rate and service time. Multiply the load by 64 kbps to estimate your demand in bits/hour.
   b. Arrival rate: Estimate the number of Web pages you request in the busiest hour of the day. Service time: Estimate the average length of a Web page. Estimate your demand in bits/hour.
   c. Compare the number of call requests/hour to the number of Web requests/hour. Comment on the connection setup capacity required if each Web page request requires a connection setup. Comment on the amount of state information required to keep track of these connections.

7. Apply the end-to-end argument to the question of how to control the delay jitter that is incurred in traversing a multihop network.

8. Compare the operation of the layer 3 entities in the end systems and in the routers inside the network.

9. In Figure 7.6 trace the transmission of IP packets from when a Web page request is made to when the Web page is received. Identify the components of the end-to-end delay.
   a. Assume that the browser is on a computer that is in the same departmental LAN as the server.
   b. Assume that the Web server is in the central organization servers.
   c. Assume that the server is located in a remote network.

10. In Figure 7.6 trace the transmission of IP packets between two personal computers running an IP telephony application. Identify the components of the end-to-end delay.
   a. Assume that the two PCs are in the same departmental LAN.
   b. Assume that the PCs are in different domains.

11. In Figure 7.6 suppose that a workstation becomes faulty and begins sending LAN frames with the broadcast address. What stations are affected by this broadcast storm? Explain why the use of broadcast packets is discouraged in IP.

12. Explain why the distance in hops from your ISP to a NAP is very important. What happens if a NAP becomes congested?

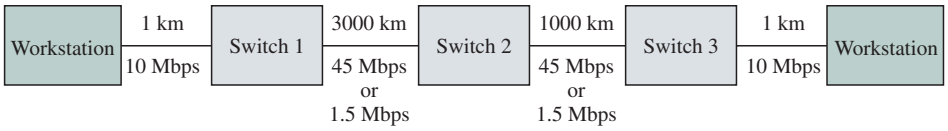
13. Consider the operation of a switch in a connectionless network. What is the source of the load on the processor? What can be done if the processor becomes the system bottleneck?

14. Consider the operation of a switch in a connection-oriented network. What is the source of the load on the processor? What can be done if the processor becomes overloaded?

15. Suppose that a computer that is used as a switch can process 20,000 packets/second. Give a range of possible bit rates that traverse the I/O bus and main memory.

16. A 64-kilobyte message is to be transmitted from over two hops in a network. The network limits packets to a maximum size of 2 kilobytes, and each packet has a 32-byte header. The transmission lines in the network are error free and have a speed of 50 Mbps. Each hop is 1000 km long. How long does it take to get the message from the source to the destination?

17. An audio-visual real-time application uses packet switching to transmit 32 kilobit/second speech and 64 kilobit/second video over the following network connection.



Two choices of packet length are being considered: In option 1 a packet contains 10 milliseconds of speech and audio information; in option 2 a packet contains 100 milliseconds of speech and audio information. Each packet has a 40 byte header.

a. For each option find out what percentage of each packet is header overhead.
b. Draw a time diagram and identify all the components of the end-to-end delay in the preceding connection. Keep in mind that a packet cannot be sent until it has been filled and that a packet cannot be relayed until it is completely received. Assume that bit errors are negligible.
c. Evaluate all the delay components for which you have been given sufficient information. Consider both choices of packet length. Assume that the signal propagates at a speed of 1 km/5 microseconds. Consider two cases of backbone network speed: 45 Mbps and 1.5 Mbps. Summarize your result for the four possible cases in a table with four entries.
d. Which of the preceding components would involve queueing delays?

18. Suppose that a site has two communication lines connecting it to a central site. One line has a speed of 64 kbps, and the other line has a speed of 384 kbps. Suppose each line is modeled by an M/M/1 queueing system with average packet delay given by $E[D] = E[X]/(1 - \rho)$ where $E[X]$ is the average time required to transmit a packet, λ is the arrival rate in packets/second, and $\rho = \lambda E[X]$ is the load. Assume packets have an average length of 8000 bits. Suppose that a fraction α of the packets are routed to the first line and the remaining $1 - \alpha$ are routed to the second line.

a. Find the value of α that minimizes the total average delay.
b. Compare the average delay in part (a) to the average delay in a single multiplexer that combines the two transmission lines into a single transmission line.

19. A message of size m bits is to be transmitted over an L-hop path in a store-and-forward packet network as a series of N consecutive packets, each containing k data bits and h header bits. Assume that $m \gg k + h$. The bit rate of each link is R bits/second. Propagation and queueing delays are negligible.
a. What is the total number of bits that must be transmitted?
b. What is the total delay experienced by the message (i.e., the time between the first transmitted bit at the sender and the last received bit at the receiver)?
c. What value of k minimizes the total delay?

20. Suppose that a datagram network has a routing algorithm that generates routing tables so that there are two disjoint paths between every source and destination that is attached to the network. Identify the benefits of this arrangement. What problems are introduced with this approach?

21. Suppose that a datagram packet network uses headers of length H bytes and that a virtual-circuit packet network uses headers of length h bytes. Use Figure 7.19 to determine the length M of a message for which virtual-circuit switching delivers the packet in less time than datagram switching does. Assume packets in both networks are the same length.

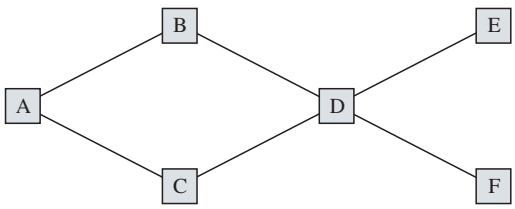
22. Suppose a routing algorithm identifies paths that are "best" in the following sense: (1) minimum number of hops, (2) minimum delay, or (3) maximum available bandwidth. Identify the conditions under which the paths produced by the different criteria are the same? are different?

23. Suppose that the virtual circuit identifiers are unique to a switch, not to an input port. What is traded off in this scenario?

24. Consider the virtual-circuit packet network in Figure 7.24. Suppose that node 4 in the network fails. Reroute the affected calls and show the new set of routing tables.

25. Consider the datagram packet network in Figure 7.26. Reconstruct the routing tables (using minimum-hop routing) that result after node 4 fails. Repeat if node 3 fails instead.

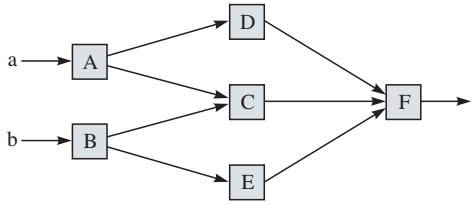
26. Consider the following six-node network. Assume all links have the same bit rate R.



a. Suppose the network uses datagram routing. Find the routing table for each node, using minimum-hop routing.

b. Explain why the routing tables in part (a) lead to inefficient use of network bandwidth.
c. Can VC routing improve efficiency in the use of network bandwidth? Explain why or why not.
d. Suggest an approach in which the routing tables in datagram routing are modified to improve efficiency. Give the modified routing tables.

27. Consider the following six-node unidirectional network. Assume all links have the same bit rate $R = 1$.



a. If flows a and b are equal, find the maximum flow that can be handled by the network.
b. If flow a is three times larger than flow b, find the maximum flow that can be handled by the network.
c. Repeat (a) and (b) if the flows are constrained to use only one path.

28. Consider the network in Figure 7.30.
a. Use the Bellman-Ford algorithm to find the set of shortest paths from all nodes to destination node 2.
b. Now continue the algorithm after the link between node 2 and 4 goes down.

29. Consider the network in Figure 7.28.
a. Use the Dijkstra algorithm to find the set of shortest paths from node 4 to other nodes.
b. Find the set of associated routing table entries.

30. Suppose that a block of user information that is L bytes long is segmented into multiple cells. Assume that each data unit can hold up to P bytes of user information, that each cell has a header that is H bytes long, and that the cells are fixed in length and padded if necessary. Define the efficiency as the ratio of the L user bytes to the total number of bytes produced by the segmentation process.
a. Find an expression for the efficiency as a function of L, H, and P. Use the ceiling function $c(x)$, which is defined as the smallest integer larger or equal to x.
b. Plot the efficiency for the following ATM parameters: $H = 5$, $P = 48$, and $L = 24k$ for $k = 0, 1, 2, 3, 4, 5,$ and 6.

31. Consider a videoconferencing application in which the encoder produces a digital stream at a bit rate of 144 kpbs. The packetization delay is defined as the delay incurred by the first byte in the packet from the instant it is produced to the instant when the packet is filled. Let P and H be defined as they are in problem 30.
a. Find an expression for the packetization delay for this video application as a function of P.
b. Find an expression for the efficiency as a function of P and H. Let $H = 5$ and plot the packetization delay and the efficiency versus P.

32. Suppose an ATM switch has 16 ports each operating at SONET OC-3 transmission rate, 155 Mbps. What is the maximum possible throughput of the switch?

33. Refer to the virtual circuit packet network in Figure 7.24. How many VCIs does each connection in the example consume? What is the effect of the length of routes on VCI consumption?

34. Generalize the hierarchical network in Figure 7.27 so that the 2^K nodes are interconnected in a full mesh at the top of the hierarchy and so that each node connects to two 2^L nodes in the next lower level in the hierarchy. Suppose there are four levels in the hierarchy.
a. How many nodes are in the hierarchy?
b. What does a routing table look like at level j in the hierarchy, $j = 1, 2, 3$, and 4?
c. What is the maximum number of hops between nodes in the network?

35. Assuming that the earth is a perfect sphere with radius 6400 km, how many bits of addressing are required to have a distinct address for every 1 cm $\times$ 1 cm square on the surface of the earth?

36. Suppose that 64 kbps PCM coded speech is packetized into a constant bit rate ATM cell stream. Assume that each cell holds 48 bytes of speech and has a 5 byte header.
a. What is the interval between production of full cells?
b. How long does it take to transmit the cell at 155 Mbps?
c. How many cells could be transmitted in this system between consecutive voice cells?

37. Suppose that 64 kbps PCM coded speech is packetized into a constant bit rate ATM cell stream. Assume that each cell holds 48 bytes of speech and has a 5 byte header. Assume that packets with silence are discarded. Assume that the duration of a period of speech activity has an exponential distribution with mean 300 ms and that the silence periods have a duration that also has an exponential distribution but with mean 600 ms. Recall that if T has an exponential distribution with mean $1/\mu$, then $P[T > t] = e^{-\mu t}$.
a. What is the peak cell rate of this system?
b. What is the distribution of the burst of packets produced during an active period?
c. What is the average rate at which cells are produced?

38. Suppose that a data source produces information according to an on/off process. When the source is on, it produces information at a constant rate of 1 Mbps; when it is off, it produces no information. Suppose that the information is packetized into an ATM cell stream. Assume that each cell holds 48 bytes of speech and has a 5 byte header. Assume that the duration of an on period has a Pareto distribution with parameter 1. Assume that the off period is also Pareto but with parameters 1 and α. If T has a Pareto distribution with parameters 1 and α, then $P[T > t] = t^{-\alpha}$ for $t > 1$. If $\alpha > 1$, then $E[T] = \alpha/(\alpha - 1)$, and if $0 < \alpha < 1$, then $E[T]$ is infinite.
a. What is the peak cell rate of this system?
b. What is the distribution of the burst packets produced during an on period?
c. What is the average rate at which cells are producd?

39. An IP packet consists of 20 bytes of header and 1500 bytes of payload. Now suppose that the packet is mapped into ATM cells that have 5 bytes of header and 48 bytes of payload. How much of the resulting cell stream is header overhead?

40. Suppose that virtual paths are set up between every pair of nodes in an ATM network. Explain why connection setup can be greatly simplified in this case.

41. Suppose that the ATM network concept is generalized so that packets can be variable in length. What features of ATM networking are retained? What features are lost?

42. Explain where priority queueing and fair queueing may be carried out in the generic switch/router in Figure 7.10.

43. Consider the head-of-line priority system in Figure 7.43. Explain the impact on the delay and loss performance of the low-priority traffic under the following conditions:
a. The high-priority traffic consists of uniformly spaced, fixed-length packets.
b. The high-priority traffic consists of uniformly spaced, variable-length packets.
c. The high-priority traffic consists of highly bursty, variable-length packets.

44. Consider the head-of-line priority system in Figure 7.43. Suppose that each priority class is divided into several subclasses with different "drop" priorities. Each priority subclass has a threshold that if exceeded by the queue length results in discarding of arriving packets from the corresponding subclass. Explain the range of delay and loss behaviors that are experienced by the different subclasses.

45. Incorporate some form of weighted fair queueing in the head-of-line priority system in Figure 7.43 so that the low-priority traffic is guaranteed to receive r bps out of the total bit rate R of the transmission link. Explain why this feature may be desirable. How does it affect the performance of the high-priority traffic?

46. Consider a packet-by-packet fair-queueing system with three logical queues and with a service rate of one unit/second. Show the sequence of transmissions for this system for the following packet arrival pattern. Queue 1: arrival at time $t = 0$, length 2; arrival at $t = 4$, length 1. Queue 2: arrival at time $t = 1$, length 3; arrival at $t = 2$, length 1. Queue 3: arrival at time $t = 3$, length 5.

47. Repeat problem 46 if queues 1, 2, and 3 have weights, 2, 3, and 5, respectively.

48. Suppose that in a packet-by-packet weighted fair-queueing system, a packet with finish tag F enters service at time t. Is it possible for a packet to arrive at the system after time t and have a finish tag less than F? If yes, give an example. If no, explain why.

49. Deficit round-robin is a scheduling scheme that operates as follows. The scheduler visits the queues in round-robin fashion. A deficit counter is maintained for each queue. When the scheduler visits a queue, the scheduler adds a quantum of service to the deficit counter, and compares the resulting value to the length of the packet at the head of the line. If the counter is larger, the packet is served and the counter is reduced by the packet length. If not, the deficit is saved for the next visit. Suppose that a system has four queues and that these contain packets of length 16, 10, 12, and 8 and that the quantum is 4 units. Show the deficit counter at each queue as a function of time and indicate when the packets are transmitted.

50. Suppose that ATM cells arrive at a leaky bucket policer at times $t = 1, 2, 3, 5, 6, 8, 11, 12, 13, 15,$ and 19. Assuming the same parameters as the example in Figure 7.55, plot the bucket content and identify any nonconforming cells. Repeat if L is reduced to 4.

51. Explain the difference between the leaky bucket traffic shaper and the token bucket traffic shaper.

52. Which of the parameters in the upper bound for the end-to-end delay (equation 11) are controllable by the application? What happens as the bit rate of the transmission links becomes very large?

53. Supose that a TCP source (with an unlimited amount of information to transmit) begins transmitting onto a link that has 1 Mbps in available bandwidth. Sketch the congestion window versus the time trajectory. Now suppose that another TCP source (also with an unlimited amount of information to transmit) begins transmitting over the same link. Sketch the congestion window versus the time for the initial source.

54. Suppose that TCP is operating in a 100 Mbps link that has no congestion.
a. Explain the behavior of slow start if the link has RTT = 20 ms, receive window of 20 kbytes, and maximum segment size of 1 kbyte.
b. What happens if the speed of the link is 1 Mbps? 100 kbps?

55. Random early detection (RED) is a buffer management mechanism that is intended to avoid congestion in a router by keeping average queue length small. The RED algorithm continuously compares a short-time average queue length with two thresholds: min_{th} and max_{th}. When the average queue length is below min_{th}, RED does not drop any packets. When the average queue length is between min_{th} and max_{th} RED drops an arriving packet with a certain probability that is an increasing function of the average queue length. The random packet drop is used to notify the sending TCP to reduce its rate before the queue becomes full. When the average queue length exceeds max_{th}, RED drops each arriving packet.
a. What impact does RED have on the tendency of TCP receivers to synchronize during congestion?
b. What is the effect of RED on network throughput?
c. Discuss the fairness of the RED algorithm with respect to flows that respond to packet drops and nonadaptive flows, for example, UDP.
d. Discuss the implementation complexity of the RED algorithm.