

FIGURE 7.20 Input buffering versus output buffering.

the first packet holding back other subsequent packets behind it is called **head-of-line (HOL) blocking**. One way to eliminate the HOL blocking is to provide N separate input buffers at each input port so that each input buffer is dedicated to a particular output. This ensures that a head-of-line packet cannot block other packets destined for different outputs. Such an input buffer is called a *virtual output buffer* since the set of virtual output buffers belonging to the same output emulates the behavior of an output buffer, and yet their physical location is at the input ports.

Recall from Chapter 4 that the complexity of a crossbar, which is N^2 , makes it undesirable for building large switches. Multistage architectures have been considered as a solution to building a large switch. One such architecture for packet switching is called a **banyan switch**, as shown in Figure 7.21. The banyan switch is typically composed of 2×2 switching elements interconnected in a certain fashion such that exactly one path exists from each input to each output. Routing can be done in a distributed manner by appending the binary address of the output number to each packet,

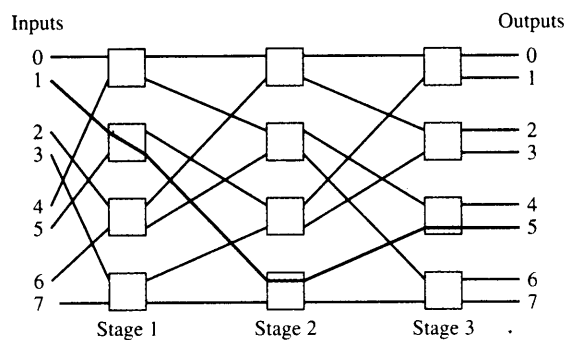


FIGURE 7.21 An 8×8 banyan switch.

and by having each switching element at stage i steer a packet based on the i th bit of the address. If the bit is 0, the switching element should steer the packet to its upper output; otherwise, steer it to the lower output. In Figure 7.21, input 1 would like to send a packet destined to output 5 (with address 101 in binary). The switching element at stage 1 looks at the first bit of the address and steers the packet to its lower output. The packet then arrives at the bottom switching element at stage 2, which steers the packet to the upper output since the second bit is 0. Finally, the switching element at stage 3 steers the packet to the lower output and sends the packet to output 5. The reader is encouraged to try this exercise with other inputs and outputs.

Notice that if there is another packet from input 5 that would like to go to output 7 at the same time, this packet will contend for the same output of the switching element at stage 1. This behavior causes the banyan switch to block packets even though some outputs are idle. One way to prevent packets from being blocked inside the switch is to provide buffering at each switching element so that contending packets may be temporarily stored at a local buffer. When the local buffer is full, the associated switching element can send a *backpressure* signal, notifying the upstream switching element to stop sending packets. The backpressure signal may be propagated all the way to the input port, which may eventually buffer the incoming packets or drop them.

The banyan switch is only one out of many possible ways to build large switches. A large literature describes how to design switch interconnection fabrics, for example, see [Robertazzi 1994].

SWITCH, ROUTER, SWITCH-ROUTER

Why are there several different names referring to a packet switch? A packet switch is often simply called a switch if connections are made before any transfer of information can take place. That is, a switch employs a connection-oriented mode and forwards a packet to its output port based on the virtual-circuit identifier of the packet. Although switches are historically implemented in hardware, it is inaccurate to refer to a device as a switch just because of implementation.

A router is a packet switch that transfers IP packets between inputs and outputs, and uses the IP address to determine the output port. Thus a router is an IP-based packet switch that operates in a connectionless mode. It is equally inaccurate to refer a device as a router if the device is implemented in software although routers were historically software-based. Indeed, most current core routers implement the forwarding path in hardware.

Recently, there is interest in combining features in connectionless and connection-oriented transfers in a single device called a switch-router that is capable of forwarding packets in dual modes. An MPLS device (discussed in Chapter 10) is one prominent example. The term “switching router” is used to emphasize the point that the device is a router that can also perform switching.

7.4 ROUTING IN PACKET NETWORKS

Routing is a major component of the network layer and is concerned with the problem of determining feasible paths (or routes) for packets to follow from each source to each destination. Figure 7.22 shows a packet-switching network providing communication services among multiple nodes.⁶ As suggested by the figure, a packet could take one of several possible paths from node 1 to node 6. 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 minimum end-to-end delay. Yet another objective function may involve selecting the path with the greatest available bandwidth. Routing algorithms 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 state of the network 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 path to the correct destination if it exists. In addition, the algorithm should not take an unreasonably long time to find the path to the destination.
2. *Adaptability to changes in network topology resulting from node or link failures.* In an operational network equipment and transmission lines are subject to failures. A routing algorithm must be able to adapt and reconfigure the paths 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 through cycles of heavy and light periods. An adaptive routing algorithm would be able to adjust the paths based on the current traffic loads.

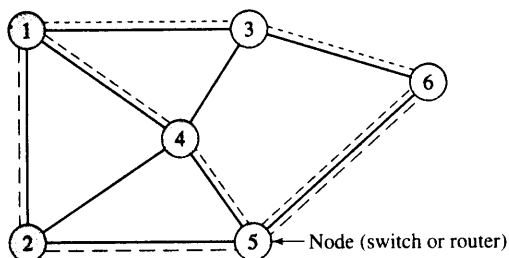


FIGURE 7.22 Multiple paths in a packet-switching network.

⁶Note that in this section a node refers to a packet switch, a router, or any network element performing routing and forwarding functions.

4. *Ability to route packets away from temporarily congested links.* A routing algorithm should avoid heavily congested links. Often it is desirable to balance the load on each link/path.
5. *Ability to determine the connectivity of the network.* To find optimal paths, the routing system needs to know the connectivity or reachability information.
6. *Ability to avoid routing loops.* Inconsistent information in distributed computation may lead to routing tables that create routing loops. The routing system should avoid persistent routing loops even in the presence of distributed routing systems.
7. *Low overhead.* A routing system typically obtains the connectivity information by exchanging control messages with other routing systems. These messages represent an overhead on bandwidth usage 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** paths are precomputed based on the network topology, link capacities, and other information. The computation is typically performed offline by a dedicated host. When the computation is completed, the paths are loaded to the routing table and remain fixed for a relatively long period of time. Static routing may suffice if the network size is small, the traffic load does not change appreciably, or the network topology is relatively fixed. Static routing may become cumbersome as the network size increases. If the traffic load changes, the pre-computed paths may easily become suboptimal. The biggest disadvantage of static routing is its inability to react rapidly to network failures. In **dynamic (adaptive) routing** each node continuously learns the state of the network by communicating with its neighbors. Thus a change in a network topology is eventually propagated to all nodes. Based on the information collected, each node can compute the best paths to desired destinations. One disadvantage of dynamic routing is the added complexity in the node.

Routing algorithms can be centralized or distributed. In **centralized routing** a network control center computes all paths and then uploads this information to the nodes in the network. In **distributed routing** nodes 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 paths calculated by different nodes are inconsistent, loops can develop. That is, if A thinks that the best path to Z is through B and B thinks that the best path 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.

Routing decisions 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 path. Datagram packet switching does not require a connection setup. The path followed by each packet is determined independently by the routing table in each node.

7.4.2 Routing Tables

Once the routing algorithm has determined the set of paths, the path information is stored in the routing table so that each node (switch or router) knows how to forward packets. As discussed in Section 7.3, the specific routing information stored depends on the type of packet switching. With virtual-circuit packet switching, the routing table translates each incoming VCI to an outgoing VCI and identifies the output port to which to forward a packet based on the incoming VCI of the packet. With datagram packet switching, the routing table identifies the next hop to which to forward a packet based on the destination address of the packet. This section describes how various routing tables in a network cooperate to carry out end-to-end forwarding of packets.

Consider a virtual-circuit packet-switching network as shown in Figure 7.23 where virtual circuits are terminated at hosts. We assume that virtual circuits are bidirectional and that each direction uses the same value. There are two virtual circuits between node A (host) and node 1 (switch). A packet sent by node A with VCI 1 in the header will eventually reach node B, while a packet with VCI 5 from node A will eventually reach node D. For each node pair, the VCI has local significance only. At each link the identifier may be translated to a different identifier, depending on the available VCIs at a given link. In our example VCI 1 from node A gets translated to 2, and then to 7, and finally to 8 before reaching node B. When node 1 receives a packet with VCI 1, that node should replace the incoming VCI with 2 and then forward the packet to node 3. Other nodes perform similarly.

Using a local VCI rather than a global one has two advantages. First, more virtual circuits can be assigned, since the VCIs 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 node has to guarantee uniqueness only on its local link—the information that the switch has in its own routing table. If global VCIs are used, the node has to ensure that the chosen VCI is not currently being used by any link along the path, a very time-consuming chore.

The corresponding routing table at each packet switch is shown in Figure 7.24. 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

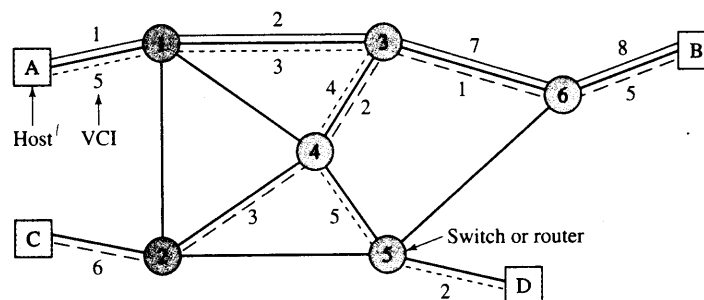


FIGURE 7.23 Virtual circuit identifier determines the destination.

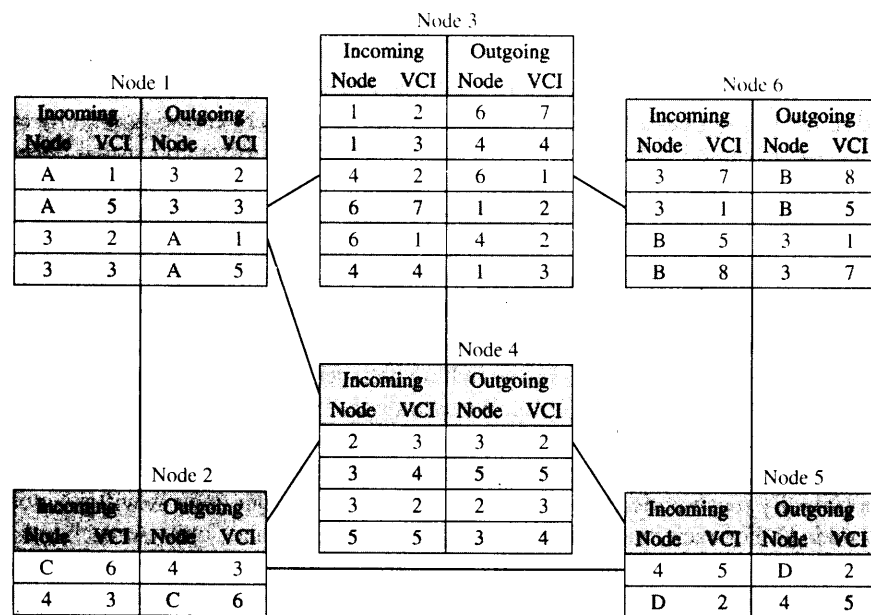


FIGURE 7.24 Routing tables for the packet-switching network in Figure 7.23.

the packet to node 5. Finally, node 5 translates the VCI to 2 and delivers the packet to the destination, which is node D. To make the description easier, the routing tables in Figure 7.24 use node numbers to identify where a packet comes from and where a packet is to be forwarded. In practice, local port numbers are used instead of remote node numbers.

With datagram packet switching, no virtual circuit has to be set up, since no connection exists between a source and a destination. Figure 7.25 shows the routing tables for the network topology in Figure 7.22, assuming that a minimum-hop routing objective 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 or more sophisticated lookup technique may be employed to yield a match quickly.

Now suppose that a packet arrives at node 1 and is destined to node 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 node D.

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 to decide how a packet should be routed. Figure 7.26 gives an example of hierarchical address assignment and a flat

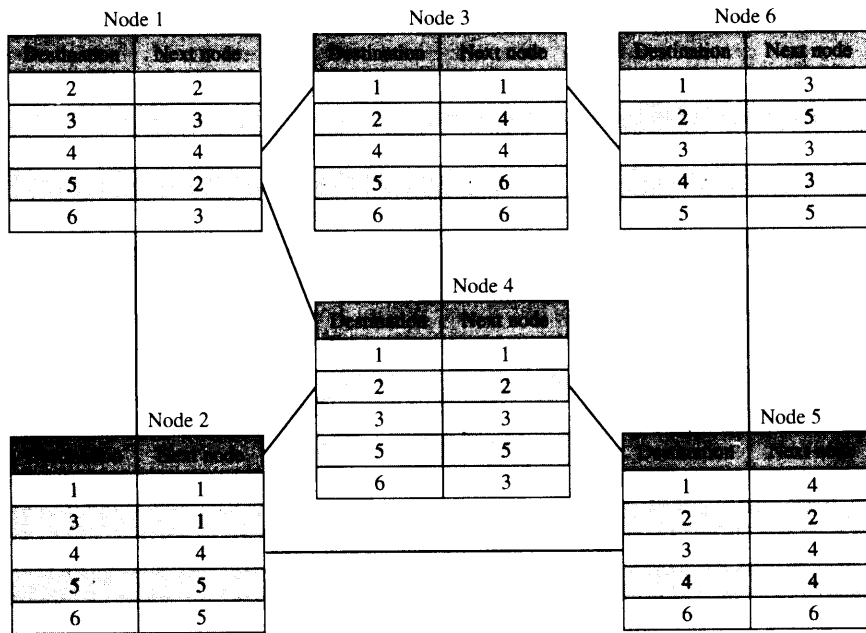


FIGURE 7.25 Routing tables for datagram network in Figure 7.22.

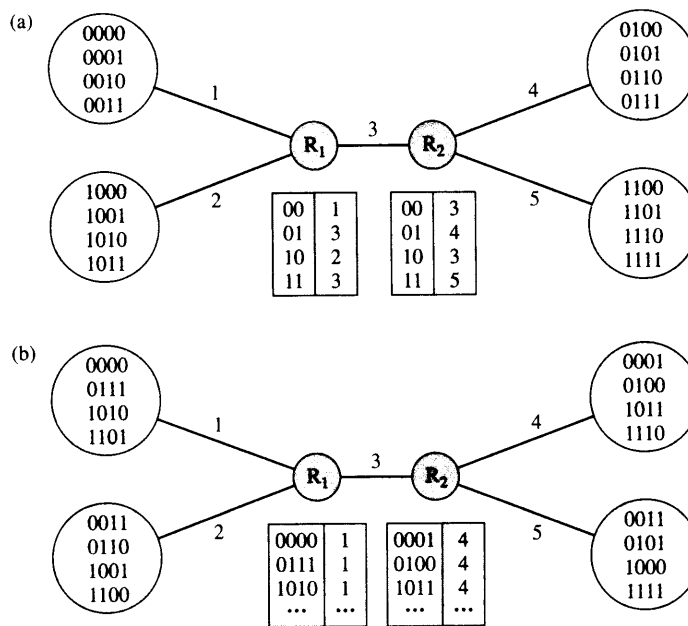


FIGURE 7.26 Address assignment: (a) hierarchical and (b) flat.

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 7.26b), then the routers need to maintain 16 entries in their routing tables.

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 network the host part of the address may be further subdivided into two parts: an identifier for a *subnetwork* within the network and a host identifier within the subnet. Outside the network, routers route packets according to the network part of the destination address. Once a packet arrives to the network, 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 a single routing table entry for the prefix. We explain the details of this procedure when we discuss CIDR addressing in Chapter 8.

7.4.4 Specialized Routing

In this section we examine two simple approaches to routing, called flooding and deflection routing, which are used in certain network scenarios.

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 packet 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 an 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 nodes connected to the network (i.e., broadcast delivery). We will see that the link-state routing algorithm uses flooding to distribute the link-state information to other nodes in the network.

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 7.27. 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 reduce resource consumption in the network, one can implement a number of mechanisms.

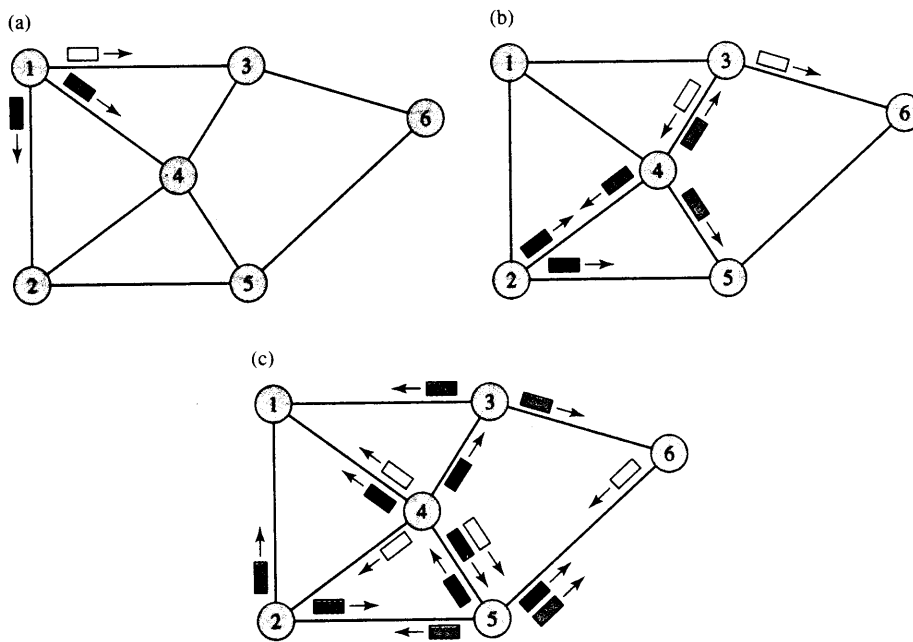


FIGURE 7.27 Flooding is initiated from node 1: (a) hop-1 transmissions, (b) hop-2 transmissions, and (c) hop-3 transmissions.

One simple method is to use a time-to-live (TTL) field in each packet. When the source sends a packet, the TTL is initially set to some number. Each node decrements the TTL by one before flooding the packet. If the value reaches zero, the node discards the packet. To avoid unnecessary waste of bandwidth, the TTL should ideally be set to the minimum hop number between two furthest nodes (called the diameter of the network). In Figure 7.27 the diameter of the network is two. To have a packet reach any destination, it is sufficient to set the TTL to two.

In the second method, each node adds its identifier to the header of the packet before it floods the packet. When a node receives a packet that contains the identifier of the node, it discards the packet since it knows that the packet already visited the node before. 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 node receives a packet, the node records the source address and the sequence number of the packet. If the node discovers that the packet has already visited the node, 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 node first tries to forward a packet

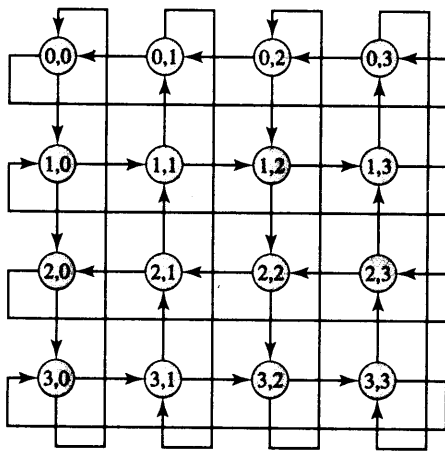


FIGURE 7.28 Manhattan street network.

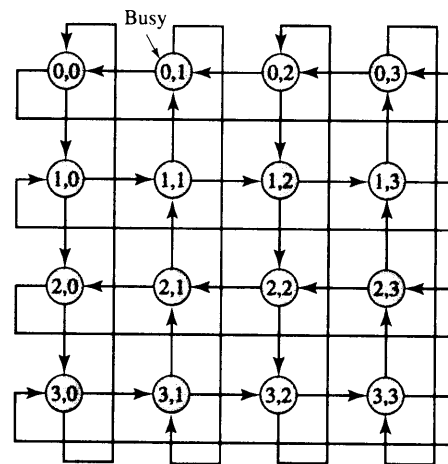


FIGURE 7.29 Deflection routing in Manhattan street network.

to the preferred port. If the preferred port is busy or congested, the packet is deflected to another port. Deflection routing often works well in a regular topology. One example of a regular topology is shown in Figure 7.28, 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 node 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 node $(0,2)$ would like to send a packet to node $(1,0)$, the packet could go two left and one down. However, if the left port of node $(0,1)$ is busy (see Figure 7.29), the packet will be deflected to node $(3,1)$. Then it can go through nodes $(2,1)$, $(1,1)$, $(1,2)$, $(1,3)$ and eventually reach the destination node $(1,0)$.

One advantage of deflection routing is that the node 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 can be useful in optical networks where optical buffers are currently expensive and difficult to build. Deflection routing can also be used to implement high-speed packet switches where the topology is usually regular and high-speed buffers are relatively expensive compared to deflection routing logic.

7.5 SHORTEST-PATH ROUTING

Many routing algorithms are based on variants of **shortest-path algorithms**, which try to determine the shortest path 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*), where each node represents a

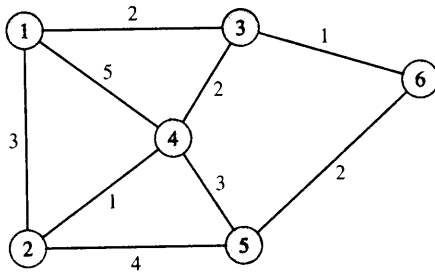


FIGURE 7.30 A network with associated link costs.

packet switch and each link represents a transmission line between two packet switches. Figure 7.30 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 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 the objective function that 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 each neighbor of node A knows the shortest path to node Z, then node A can determine its shortest path to node Z by calculating the cost/distance to node Z through each of its neighbors and picking the minimum.

As an example, suppose that we want to find the shortest path from node 2 to node 6 (the destination) in Figure 7.30. 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, $C_{13} = C_{31} = 2$, and $C_{45} = 3$ in Figure 7.30. 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 7.30. If, in Figure 7.30, the destination node is node 6, then the minimum cost from node 2 to the destination node 6 can be calculated in terms of distances through node 1, node 4, or node 5:

$$\begin{aligned} D_2 &= \min\{C_{21} + D_1, C_{24} + D_4, C_{25} + D_5\} \\ &= \min\{3 + 3, 1 + 3, 4 + 2\} \\ &= 4 \end{aligned} \tag{7.5}$$

Thus the minimum cost from node 2 to node 6 is through node 4 and is equal to 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, the cost from node 1 to the destination node 6 is found from

$$D_1 = \min\{C_{12} + D_2, C_{13} + D_3, C_{14} + D_4\} \tag{7.6}$$

and similarly the cost from node 4 is found from

$$D_4 = \min\{C_{41} + D_1, C_{42} + D_2, C_{43} + D_3, C_{45} + D_5\} \tag{7.7}$$

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 $D_1 = D_2 = \dots = D_5 = \infty$. Observe that at each iteration we may discover new shorter paths to the destination, and so the distances from each node to the destination node 6, that is, D_1, D_2, \dots, D_5 are nonincreasing. Because the minimum distances are bounded below, eventually D_1, D_2, \dots, D_5 must converge to the distances corresponding to the shortest path to the given destination.

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

1. Initialization (destination node d is distance 0 from itself)

$$D_i = \infty, \quad \text{for all } i \neq d \tag{7.8}$$

$$D_d = 0 \tag{7.9}$$

2. Updating (find minimum distance to destination through neighbors): For each $i \neq d$,

$$D_i = \min_j \{C_{ij} + D_j\}, \quad \text{for all } j \neq i \quad (7.10)$$

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

EXAMPLE Minimum Cost

Using Figure 7.30, apply the Bellman-Ford algorithm to find both the minimum cost from each node to the destination (node 6) and the next node along the shortest path.

Each node i maintains an entry (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 given by the value of j in Equation 7.10, which gives the minimum cost. If the next node is not defined, we set n to -1 . In the first iteration, the destination node informs its directly-attached neighbors that it is distance zero from itself. This prompts the neighbors to calculate their distance to the destination node. In iteration 2, the directly-attached neighbors inform their neighbors of their current shortest distance to the destination, so all nodes within two hops of the destination have found a path to the destination. At iteration m , all nodes within m hops of the destination node have determined a path to the destination. The algorithm terminates when no more changes in entries are observed. Table 7.2 shows the execution of the Bellman-Ford algorithm for destination node 6.

- Initially all nodes, other than the destination node 6, are at infinite cost (distance) to node 6. Node 6 informs its neighbors it is distance 0 from itself.
- (Iteration 1) Node 3 finds that it is connected to node 6 with cost of 1. Node 5 finds it is connected to node 6 at a cost of 2. Nodes 3 and 5 update their entries and inform their neighbors.
- (Iteration 2) Node 1 finds it can reach node 6, via node 3 with cost 3. Node 2 finds it can reach node 6, via node 5 with cost 6. Node 4 finds it has paths via nodes 3 and 5, with costs 3 and 7 respectively. Node 4 selects the path via node 3. Nodes 1, 2, and 4 update their entries and inform their neighbors.
- (Iteration 3) Node 2 finds that it can reach node 6 via node 4 with distance 4. Node 2 changes its entry to (4, 4) and informs its neighbors.
- (Iteration 4) Nodes 1, 4, and 5 process the new entry from node 2 but do not find any new shortest paths. The algorithm has converged.

TABLE 7.2 Sample processing of Bellman-Ford algorithm. Each entry for node i represents the next node and cost of the current shortest path to destination 6.

Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
Initial	(-1, ∞)	(-1, ∞)	(-1, ∞)	(-1, ∞)	(-1, ∞)
1	(-1, ∞)	(-1, ∞)	(6, 1)	(-1, ∞)	(6, 2)
2	(3, 3)	(5, 6)	(6, 1)	(3, 3)	(6, 2)
3	(3, 3)	(4, 4)	(6, 1)	(3, 3)	(6, 2)
4	(3, 3)	(4, 4)	(6, 1)	(3, 3)	(6, 2)

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 7.2, 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 7.31 shows the shortest-path tree rooted to node 6.

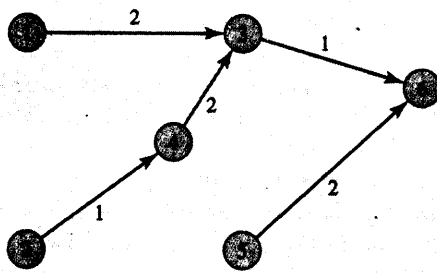


FIGURE 7.31 Shortest-path tree to node 6.

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 should 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 protocol implementing the distributed Bellman-Ford algorithm is often referred to as a *distance vector protocol*. Each node i participating in the distance vector protocol computes the following equation:

$$D_{ii} = 0 \quad (7.11)$$

$$D_{ij} = \min_k \{C_{ik} + D_{kj}\}, \quad \text{for all } k \neq i \quad (7.12)$$

where D_{ij} is the minimum cost from node i to the destination node j . Upon updating, node i broadcasts the vector $\{D_{i1}, D_{i2}, D_{i3}, \dots\}$ 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 7.30, 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 7.32.

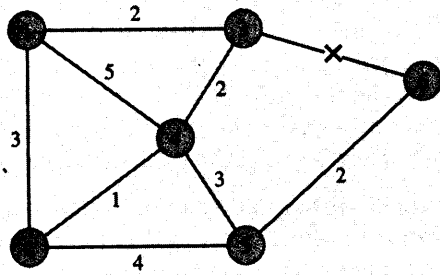


FIGURE 7.32 New network topology following break from node 3 to 6.

Upon receiving routing updates, we assume that nodes recompute their shortest paths in parallel simultaneously. When the computations are completed, we also assume that routing updates are transmitted simultaneously. The results of the computations are shown in Table 7.3. The results are obtained as follows (again the reader is encouraged to perform the algorithm before reading the discussion).

- (Update 1) As soon as node 3 detects that link (3,6) breaks, node 3 recomputes the minimum cost to node 6. Node 3 looks for paths to node 6 through its neighbors, node 1 and node 4, and its calculations indicate that the new shortest path is through node 4 at a cost of 5. (At this point, node 3 does not realize that the shortest path that node 4 is advertising happens to go through node 3, so a loop has been created!) Node 3 then sends the new routing update to its neighbors, which are nodes 1 and 4.
- (Update 2) Nodes 1 and 4 then recompute their minimum costs: node 1 finds its shortest path is still through node 3 but the cost has increased to 7; node 4 finds its shortest path is through either node 2 or node 5 with a cost of 5. We suppose that node 4 chooses node 2 (which creates a new loop between 4 and 2). Node 1 transmits its routing update to nodes 2, 3, and 4, and node 4 transmits its routing update to nodes 1, 2, 3, and 5.
- (Update 3) Node 1 finds its shortest path is still through node 3. Node 2 finds its shortest path is still through node 4 but the cost has increased to 6. Node 3 finds that its shortest path is still through node 4 but the cost has increased to 7. Nodes 4 and 5 find their shortest paths have not changed. Node 2 transmits its update to nodes 1, 4, and 5, and node 3 transmits its update to nodes 1 and 4.

TABLE 7.3 Next node and cost of current shortest path to node 6 using the distributed version.

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)

- (Update 4) Node 1 finds its shortest path is through either node 2 or node 3 with a cost of 9. Suppose that node 1 chooses node 2. Node 4 now finds a new shortest path through node 5 with a cost of 5 (which removes the loop), since the cost through 2 has increased to 7. Node 5 does not change its shortest path. Node 1 transmits its update to nodes 2, 3, and 4, and node 4 transmits its update to nodes 1, 2, 3, and 5.
- (Update 5) None of the nodes finds a new shorter path. The algorithm has converged.

This example shows that the distributed version of the Bellman-Ford algorithm requires a series of exchanges of update information to correct inaccurate information until convergence is achieved. Because of this, the algorithm usually converges rather slowly. Note also that during the calculation in the transient state, packets already in transit may loop among nodes. After convergence in the steady state, packets eventually find the destination.

EXAMPLE Reaction to Link Failure

This example shows that the distributed Bellman-Ford algorithm may react very slowly to a link failure. To see this, consider the topology shown in Figure 7.33a with node 4 as the destination. Suppose that after the algorithm stabilizes, link (3,4) breaks, as shown in Figure 7.33b. Recompute the minimum cost from each node to the destination node (node 4).

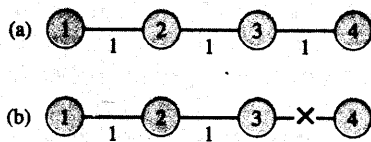


FIGURE 7.33 Topology before and after link failure.

TABLE 7.4 Next node and cost of current shortest path to node 4.

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

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

The computation of minimum costs is shown in Table 7.4. As the table shows, each node keeps updating its cost (in increments of 2 units). At each update, 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 7.33a. Suppose that after the algorithm 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 7.5. 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.

TABLE 7.5 Minimum costs by using split horizon with poisoned reverse.

Update	Node 1	Node 2	Node 3
Before break	(2, 3)	(3, 2)	(4, 1)
After break	(2, 3)	(3, 2)	(-1, ∞)
1	(2, 3)	(-1, ∞)	(-1, ∞)
2	(-1, ∞)	(-1, ∞)	(-1, ∞)

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 progressively identify 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 source node or the first two closest nodes, and so on. Thus at the k th 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 N and the distance to the remaining nodes via the new node is evaluated. 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:

$$N = \{s\} \quad (7.13)$$

$$D_j = C_{sj}, \quad \text{for all } j \neq s \quad (7.14)$$

$$D_s = 0 \quad (7.15)$$

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

$$D_i = \min_{j \notin N} D_j \quad (7.16)$$

Add i to N .

If N contains all the nodes, stop.

3. Updating minimum costs after node i added to N : For each node $j \notin N$

$$D_j = \min\{D_j, D_i + C_{ij}\} \quad (7.17)$$

Go to step 2.

EXAMPLE Finding the Shortest Path

Using Figure 7.30, apply Dijkstra's algorithm to find the shortest paths from the source node (assumed to be node 1) to all the other nodes.

Table 7.6 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 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.

TABLE 7.6 Execution of Dijkstra's algorithm.

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

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 7.34. When the algorithm stops, it 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 7.7.

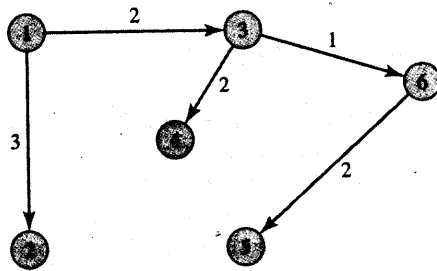


FIGURE 7.34 Shortest-path tree from node 1 to other nodes.

The data in the table is obtained as follows (again we encourage the reader to develop the results in Table 7.6 and Table 7.7, before reading on):

- (Iteration 1) Node 1 compares its costs to its directly attached nodes and finds that node 3 is the closest with a cost of 2. Node 3 is added to the set N . The tree diagram leading to Figure 7.34 is started by linking node 1 to node 3. The new minimum costs from node 1 to other nodes via node 3 are determined. It is found that a new shortest path to node 4 is through node 3 with cost of 4 so the third entry in iteration 1 is changed. It is also determined that node 6 can be reached through node 3 at a cost of 3.
- (Iteration 2) Node 2 and node 6 are tied for the second closest from node 1. Suppose that node 2 is selected so it is added to the set N . The tree diagram is updated by

connecting node 1 to node 2. The new minimum costs from node 1 are now calculated for the paths through node 2. It is found that node 5 has a cost of 7 through node 2.

- (Iteration 3) Node 6 is next added to N and connected in the tree diagram to node 1 via node 3. A new shortest path to node 5 through node 6 is found with a cost of 5.
- (Iteration 4) Node 4 is found to be the fourth closest node from node 1. It is connected to node 1 via node 3. No new shortest path is found through node 4.
- (Iteration 5) Node 5 is finally added to N with a cost of 5. Node 5 is connected to node 1 via nodes 3 and 6. This completes the algorithm.

TABLE 7.7 Routing table at node 1 for Figure 7.34.

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

To calculate the shortest paths, Dijkstra's 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 *link-state protocol* uses this approach to calculate shortest paths.

7.5.3 Source Routing versus Hop-by-Hop Routing

In the datagram network, typically each node is responsible for determining the next hop along the shortest path. If each node along the path performs the same process, a packet traveling from the source is said to follow **hop-by-hop routing** to the destination.

Source routing is another routing approach whereby the path to the destination is determined by the source. Another variation, termed **explicit routing**, allows a particular node (not necessarily the source) to determine the path. Source routing works in either datagram or virtual-circuit packet switching. Before the source can send a packet, the source has to know the path to the destination in order to include the path information in the packet header. The path 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 7.35 shows how source routing works in a datagram network.

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

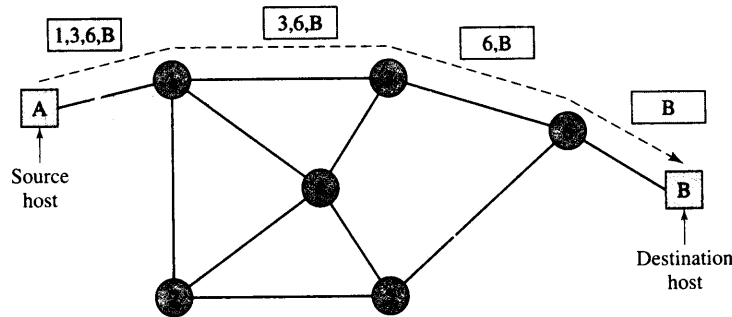


FIGURE 7.35 Example of source routing.

In some cases it may be useful to preserve the complete path information while the packet is progressing toward the destination. With complete path information the destination can send a packet back to the source by simply reversing the path, which avoids relearning the reverse path. Path preservation can easily be implemented by introducing another field in the header that keeps track of the next node to be visited along the path so that a node knows which specific address to read.

Source routing typically can be *strict* or *loose*. In strict source routing the source specifies the address of each node along the path to the destination. For cases when the source only knows partial information of the network topology, the source can use loose source routing where only a subset of the nodes along the path needs to be specified. For example, host A may specify path (1, 6, B) for loose source routing in Figure 7.35. When node 1 receives the packet containing such a path, it is up to node 1 to determine the path to node 6. For example, node 1 may decide that the better path to node 6 is through node 4 and node 5.

7.5.4 Link-State Routing versus Distance-Vector Routing

Within a domain the 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, and available bandwidth. To perform the shortest-path calculations, the values of the metrics for different links in the networks are required. Nodes must cooperate and exchange information to obtain the values of the metrics. They then use one of the two types of shortest-path algorithms to compute the set of current best routes.

Many routers in the Internet support both the **distance-vector protocol** and the **link-state protocol**. 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 using a Bellman-Ford algorithm to see whether they can find new better paths through the neighbor that provided the information. If a new better path is found, the router will send the new vector to its neighbors. Distance-vector routing adapts 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 process allows each router to construct a map of the entire network and from this map to derive the routing table using the Dijkstra algorithm. If the state of the link changes, the router detecting the change will flood the new information throughout the network. Thus link-state routing typically converges faster than distance-vector routing.

Besides convergence time, comparison between distance vector and link-state routing has been made in the past with respect to memory and processing requirements. However, these criteria have become less important due to rapid advances in silicon technologies that have made these components significantly cheaper and more powerful.

Since link-state routing knows the entire map of the network, link-state routing is generally more flexible than distance-vector routing. For example, link-state routing deals with source routing better than distance-vector routing. Some networks may impose certain constraints for a given source-destination pair, such as avoiding a particular link that is deemed unreliable. In such cases, link-state routing also works better than distance-vector 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 can provide quality-of-service (QoS) guarantees on a per-connection basis.

ATM combines several desirable features of packet switching and time-division multiplexing (TDM) circuit switching. Table 7.8 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 i.e., video, 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 significant difficulty supporting variable-bit-rate services. Because TDM systems inherently 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 and STS-1 for SONET.

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.

TABLE 7.8 TDM versus packet multiplexing.

	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

Packet multiplexing, on the other hand, has inherently variable transfer delays because of the queuing that takes place in the multiplexers. Packet multiplexing may also have 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. ATM avoids this situation by making the packet length short.

The third criterion for comparison is the capability to support bursty traffic. TDM dedicates the transmission resources, namely, slots, to a connection for the entire duration of the 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, the processing in transferring slots from inputs to outputs is relatively small and can be done at very high speeds. Packet multiplexing, on the other hand, has to examine the information in each packet header and requires more intensive processing. Packet processing was traditionally done in software, which was relatively slow at the time ATM was first formulated. However, recent advances in hardware techniques have opened up opportunities for utilizing hardware to perform intensive packet processing at high speed.

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 53-byte fixed-length packets called **cells**. Cells contain abbreviated 5-byte 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 7.36 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

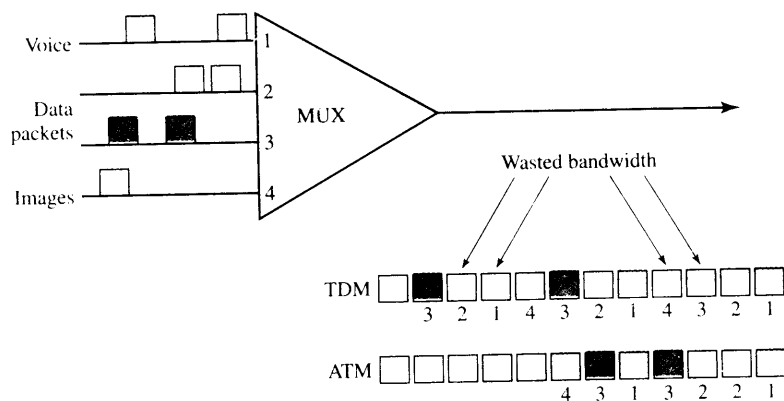


FIGURE 7.36 ATM multiplexing.

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-switching networks. ATM 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 QoS parameters that the connection must satisfy, 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. Figure 7.37 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 high speed operation possible. Indeed, ATM switches with capacities of up to 640 Gigabits/second have been deployed in the field. The use of short fixed-length packets also gives a finer degree

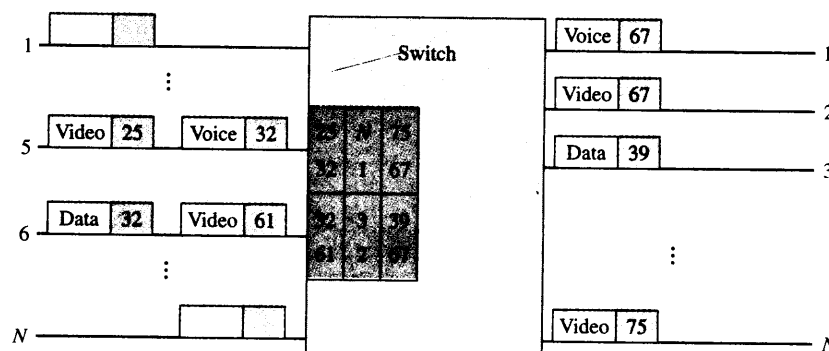


FIGURE 7.37 ATM switching.

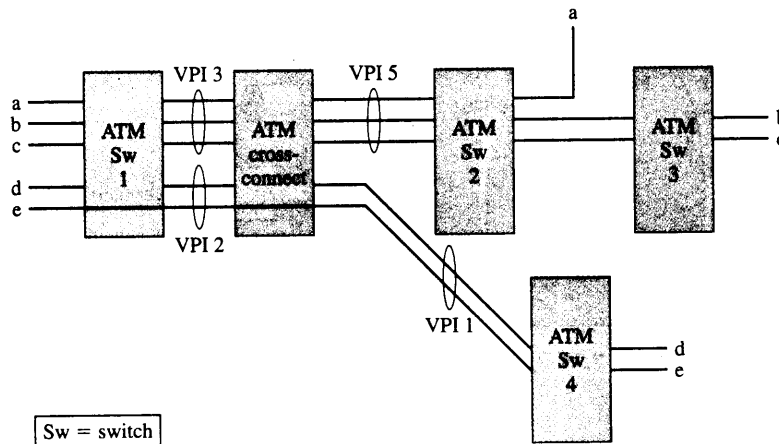


FIGURE 7.38 Role of virtual paths in ATM network.

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 7.38 shows five VCCs in an ATM network. The VCCs a, b, and c enter the network at switch 1, share a common path up to switch 2, and are bundled into an aggregated path called a **virtual path connection (VPC)** that connects switch 1 to switch 2.⁷ This VPC happens to pass through an ATM cross-connect whose role 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 7.38 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)**. A VPC can be thought of as a large digital pipe that takes traffic streams from other smaller pipes (VCCs) and handles them as a single unit. Figure 7.39 shows a cross-section of the cell stream that arrives

⁷We 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.

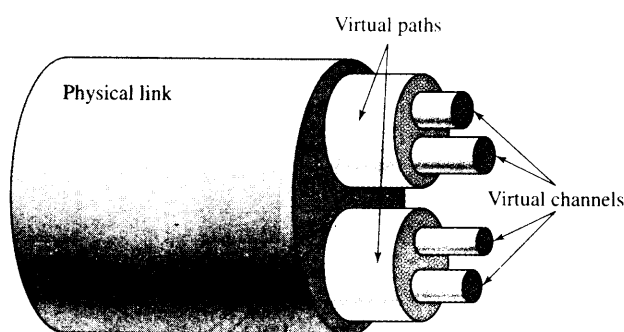


FIGURE 7.39 ATM virtual connections.

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.

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-connects allow 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.

FROM ATM TO MPLS

ATM was initially envisioned as the high-speed multiservice network technology that could be deployed in the core packet network. In the 1990s, ATM switches were widely used to interconnect IP routers in the ISP network. The connection-oriented feature of ATM allows service providers to easily configure the routes followed by the virtual circuits and paths so that network congestion is minimized. However, the overhead in the ATM cell header was found to be a significant penalty in link efficiency. The need to implement segmentation and reassembly for ATM at speeds beyond OC-48 also proved challenging and unnecessary given the much lower packet transmission times inherent at such high transmission speeds.

Multi-Protocol Label Switching (MPLS) was introduced as an alternative network technology for the core ISP network. As in ATM, MPLS adopts the label switching paradigm, but with variable-length packets using Packet-over-SONET (POS) encapsulation (MPLS is described in detail in Chapter 10). This difference avoids the inefficiency of ATM overhead and the need to do segmentation and reassembly. In addition, MPLS integrates its routing and addressing functions with IP (unlike ATM which requires separate signaling and routing functions) and so is better suited for networks that primarily carry IP. At present, ATM is mainly used in the edge of the network where QoS is essential and speed is not of essence.

7.7 TRAFFIC MANAGEMENT AT THE PACKET LEVEL

Traffic management is concerned with delivery of QoS to the end user and with efficient use of network resources. Based on traffic granularity, we can classify traffic management into three levels: packet level, flow level, and flow-aggregated level. In this section, we focus on the packet level, which is mainly concerned with packet queuing and packet scheduling at switches, routers, and multiplexers to provide differentiated treatment for packets belonging to different QoS classes. Packet-level traffic management operates in the smallest time scale on the order of packet transmission time.

Recall from Section 7.3.3 that a packet switch can be viewed as a node where packet streams arrive, are demultiplexed, switched, and remultiplexed onto outgoing lines. A packet switch also contains buffers to ensure that coincident packet arrivals are not lost. Thus the path traversed by a packet through a network can be modeled as a sequence of queuing systems as shown in Figure 7.40. 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 node and depart at some later node, 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 queuing systems. For example, the total *end-to-end delay* is the sum of the individual delays experienced at each system. Therefore,

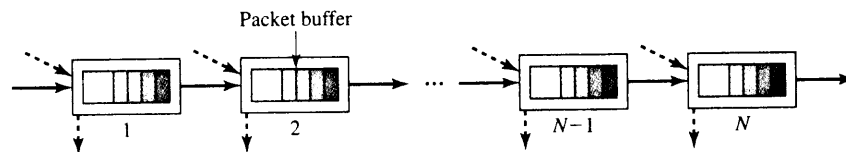


FIGURE 7.40 The end-to-end QoS of a packet along a path traversing N queuing systems.

the average end-to-end delay is the sum of the individual average delays. If we can guarantee that the delay at each system can be kept below some upper bound, then the end-to-end delay can be kept below the sum of the upper bounds. 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 queueing system that has no more buffers available. Causes of packet loss include surges in packet arrivals to a buffer and increased transmission time due to long packets 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 system.

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.⁸ 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, a queueing system must implement strategies for controlling the transmission bit rates that are provided to the various information flows (called **queue scheduling**), and for managing how packets are placed in the queueing system (called **queue management**). We now consider a number of these strategies.

7.7.1 FIFO and Priority Queues

The simplest approach to queue scheduling involves **first-in, first-out (FIFO) queueing** where packets are transmitted in order of their arrival, as shown in Figure 7.41a. Packets are discarded when they arrive at a full buffer. The delay and loss experienced by packets in a FIFO queueing system depend on the packet interarrival times and on the packet lengths. As interarrivals become more bursty or packet lengths more variable, packets will tend to bunch up and queues will then build up, causing performance to 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 queueing 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 buffer.

The FIFO queue management can be modified to provide different characteristics of packet-loss performance to different classes of traffic. Figure 7.41b shows an example

⁸For example, in IP networks the path between a source and destination remains fixed once routing updates are processed and routing tables updated.

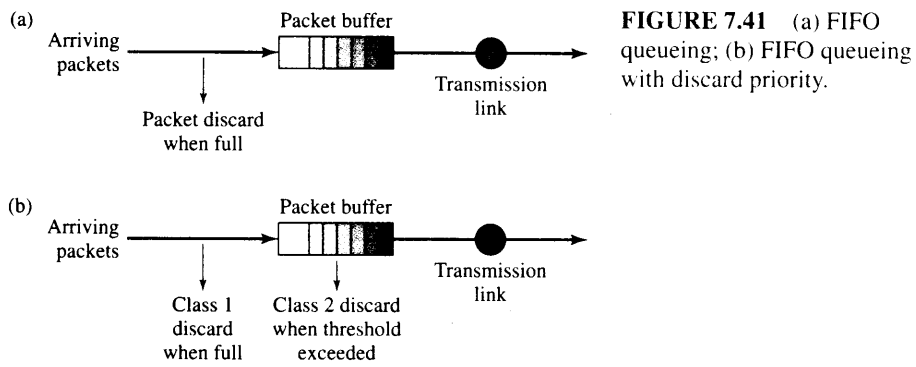


FIGURE 7.41 (a) FIFO queueing; (b) FIFO queueing with discard priority.

with two classes of traffic. When the number of packets in a buffer 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 queue scheduling approach that involves defining a number of priority classes. A separate buffer is maintained for each priority class. As shown in Figure 7.42, each time the transmission link 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 degree of guaranteed access to transmission bandwidth to the lower priority classes. Another problem is that it does not discriminate among 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 queue scheduling, shown in Figure 7.43, involves sorting packets in the buffer 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

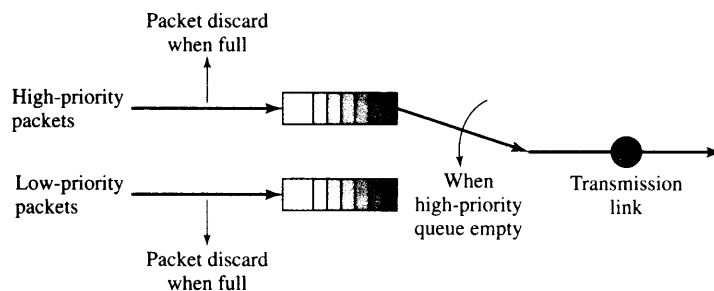


FIGURE 7.42 HOL priority queueing.

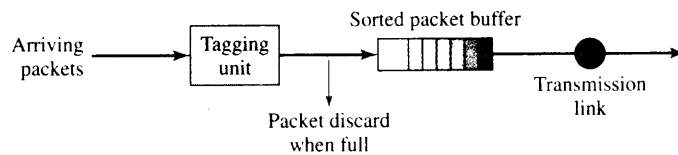


FIGURE 7.43 Sorting packets according to priority tag.

defining priority is open and can even be defined dynamically.⁹ For example, the priority tag could consist of a priority class followed by the arrival time of a packet. The resulting system implements the HOL priority system discussed above. In a second example the priority tag corresponds to a due date. Packets requiring smaller delays are assigned earlier due dates and are transmitted sooner. 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.

7.7.2 Fair Queueing

Fair queueing attempts to provide equitable access to transmission bandwidth. Each user flow has its own logical buffer. In an ideal system the transmission bandwidth, say, C bits/second, is divided equally among the buffers that have packets to transmit.¹⁰ The contents of each buffer can then be viewed as a *fluid* that is drained continuously. 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 buffers. 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 buffers will be empty from time to time, so a share larger than C/n bits/second is received at those times.

In practice, dividing the transmission capacity exactly equally is not possible. As shown in Figure 7.44 one approach could be to service each nonempty buffer 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 output. 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 system needs only service the nonempty buffers one packet at a time in round-robin fashion. User flows are then guaranteed equal access to the transmission bandwidth.

⁹See [Hashemi 1997] for a discussion on the various types of scheduling schemes that can be implemented by this approach.

¹⁰This technique is called processor sharing in the computing literature.

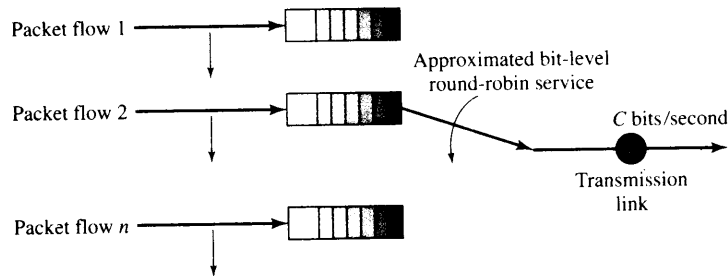


FIGURE 7.44 Fair queueing.

Figure 7.45 illustrates the differences between ideal or “fluid flow” and packet-by-packet fair queueing. The figure assumes that buffer 1 and buffer 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 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 buffer 1, followed by one bit from buffer 2, and so on. After the first bit each subsequent bit from buffer 1 would require $2/L$ seconds to transmit. Therefore, the transmission of the packet from buffer 1 would be completed after $1 + 2(L - 1) = 2L - 1$ bit-transmission times, which equals $2 - 1/L$ seconds. The packet from buffer 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 buffer 1 first

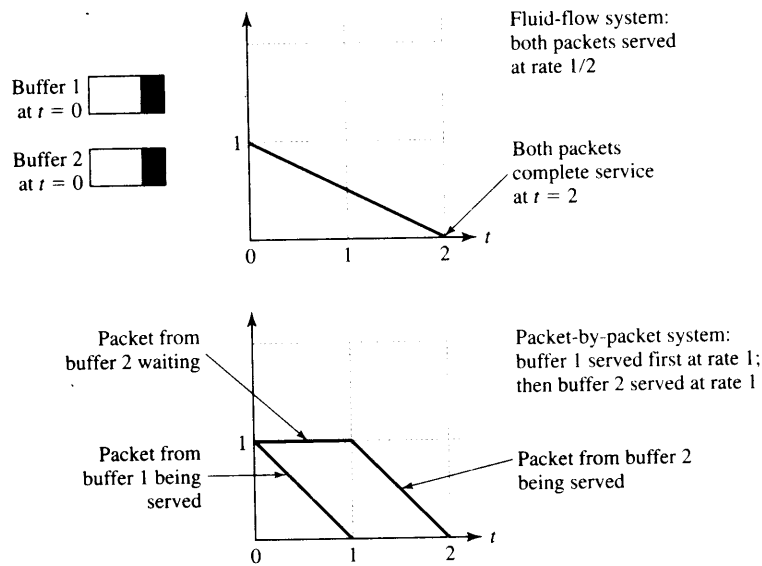


FIGURE 7.45 Fluid-flow and packet-by-packet fair queueing (two packets of equal length).

and then transmits the packet from buffer 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 queuing is not as straightforward when packets have variable lengths. If the different user buffers 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 buffers so that the packet completion times approximate those of a fluid-flow fair-queueing system. Each time a packet arrives at a user buffer, 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 buffers. We refer to this system as a **packet-by-packet fair-queueing** system.

Consider how a finish tag may be computed. Assume that there are n flows, each with its own buffer. Suppose for now that each buffer is served one bit at a time. Let a *round* consist of a cycle in which all n buffers are offered service as shown in Figure 7.46. The number of rounds in a time period is the number of opportunities that each buffer has had to transmit a bit. If a packet of length k bits were to begin transmission at a given point in time, then its packet transmission would be completed k rounds into the future. Thus the notion of rounds can be related to relative packet completion times. The actual duration of a given round is proportional to the actual number of buffers $n_{active}(t)$ that have information to transmit. When the number of active buffers is large, the duration of a round is large; when the number of active buffers is small, the rounds are short in duration.

Now suppose that the buffers 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 buffers. Each time $R(t)$ reaches a new integer value marks an instant at which all the buffers have been given an equal number of opportunities to transmit a bit. Because of the fluid-flow assumption, $R(t)$ is a continuous function that increases at a rate that is inversely proportional to the number of active buffers; that is,

$$dR(t)/dt = C/n_{active}(t) \tag{7.18}$$

Number of rounds = Number of bit transmission opportunities

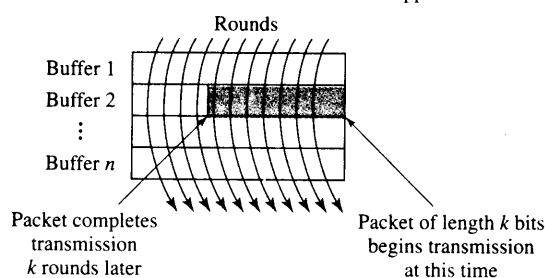


FIGURE 7.46 Computing the finishing time in packet-by-packet fair queueing and weighted fair queueing.

where C is the transmission capacity. Note that the slope of $R(t)$ changes each time the number of active buffers changes, so $R(t)$ is a piecewise linear function.

Let us see how we can calculate the finish tags to approximate fluid-flow fair queueing. Suppose that the k th packet from flow i arrives at an empty buffer 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:

$$F(i, k) = R(t_k^i) + P(i, k) \quad (7.19)$$

We will use $F(i, k)$ as the finish tag of the packet. On the other hand, if the k th packet from the i th flow arrives at a nonempty buffer, 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,

$$F(i, k) = F(i, k - 1) + P(i, k) \quad (7.20)$$

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

$$F(i, k) = \max\{F(i, k - 1), R(t_k^i)\} + P(i, k) \quad \text{for fair queueing.} \quad (7.21)$$

We reiterate: The actual packet completion time for the k th 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 buffers. However, the order in which packets from all flows complete their transmissions follows $F(i, k)$. Therefore the finish tags can be used as priorities in a packet-by-packet system: Each time a packet transmission is completed, the next packet to be transmitted is the one with the smallest finish tag.

As an example, suppose that at time $t = 0$ buffer 1 has one packet of length one unit and buffer 2 has one packet of length two units. A fluid-flow system services each buffer at rate $1/2$ as long as both buffers remain nonempty. As shown in Figure 7.47, buffer 1 empties at time $t = 2$. Thereafter buffer 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 buffer 1 is $F(1, 1) = R(0) + 1 = 1$. The finish tag of the packet from buffer 2 is $F(2, 1) = R(0) + 2 = 2$. Since the finish tag of the packet of buffer 1 is smaller than the finish tag of buffer 2, the system will service buffer 1 first. Thus the packet of buffer 1 completes its transmissions at time $t = 1$ and the packet of buffer 2 completes its transmissions at $t = 3$.

7.7.3 Weighted Fair Queueing

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

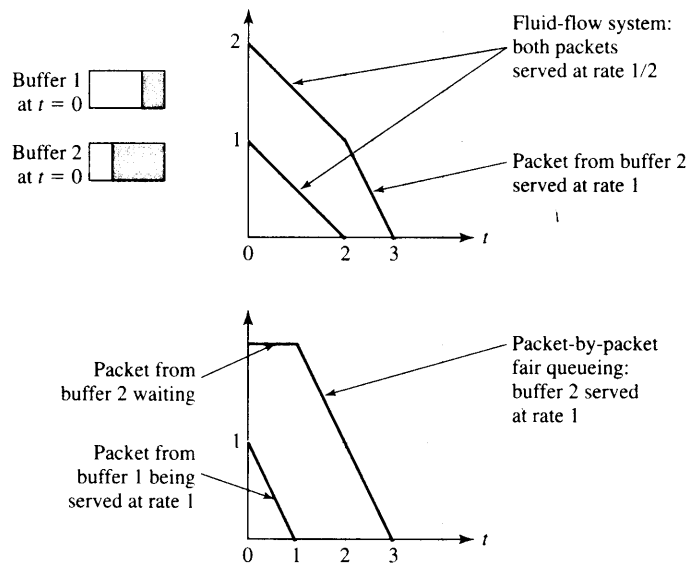


FIGURE 7.47 Fluid-flow and packet-by-packet fair queuing (two packets of different lengths).

Figure 7.48 shows the completion times for the fluid-flow case where both buffers have a one-unit length packet at time $t = 0$. For the fluid-flow system, the transmission of the packet from buffer 2 is completed at time $t = 4/3$, and the packet from buffer 1 is completed at $t = 2$. The bit-by-bit approximation to weighted fair queuing would operate by allotting each buffer a different number of bits/round. In the preceding example, buffer 1 would receive 1 bit/round and buffer 2 would receive 3 bits/round.

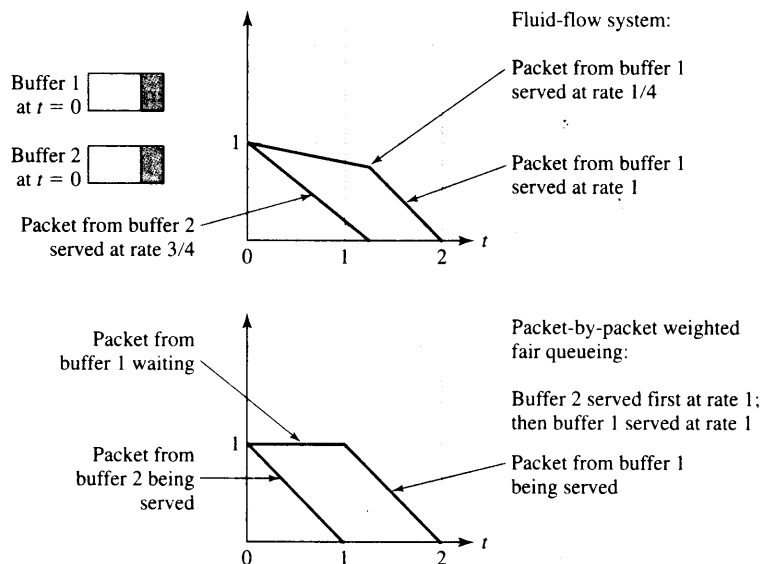


FIGURE 7.48 Fluid-flow and packet-by-packet weighted fair queuing.

PROVIDING QoS IN THE INTERNET

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. Packets are marked at the edge of the network to indicate the type of treatment that they are to receive in the routers inside the network. This approach does not provide strict QoS guarantees. 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 is also easily approximated in ATM: In each round each nonempty buffer 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:

$$F(i, k) = \max\{F(i, k-1), R(t_k^i)\} + P(i, k)/w_i \quad \text{for weighted fair queueing.} \quad (7.22)$$

Thus from the last term in Equation 7.22, 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 is calculated assuming a depletion rate that is twice that of a packet from flow j .

Figure 7.48 also shows the completion times for the packet-by-packet weighted fair-queueing system. The finish tag of the packet from buffer 1 is $F(1, 1) = R(0) + 1/1 = 1$. The finish tag of the packet from buffer 2 is $F(2, 1) = R(0) + 1/3 = 1/3$. Therefore the packet from buffer 2 is served first. The packet for buffer 2 is now completed at time $t = 1$, and the packet from buffer 1 at time $t = 2$. Note that packet-by-packet weighted fair queueing is also applicable when packets are of different length.

Weighted fair-queueing systems are a means for providing QoS guarantees. Suppose that 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 buffers are nonempty, the given user flow will receive a fraction w_i/W of the bandwidth C . When other user buffers are empty, the given user flow will receive a greater share. Thus the user is guaranteed a minimum long-term bandwidth of at least $(w_i/W)C$ bits/second. This guaranteed share of the bandwidth to a large extent insulates the given user flow from the other user flows.

In addition, if the user information arrival rate is *regulated* to satisfy certain conditions, then the maximum delay experienced in the queueing system 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 queueing systems that use packet-by-packet weighted fair queueing. These bounds depend on the maximum burst that the user is allowed to submit at each queueing system, on the weights at the various queueing

systems, and on the maximum packet size that is allowed in the network. We return to the details of this scheme in Section 7.8.

7.7.4 Random Early Detection

Although fair queueing provides a fair allocation of bandwidth among multiple flows, it is rarely implemented in a core network where tens of thousands of flows may be present at any given time. Maintaining a large number of flows requires the tracking of many states, which can increase the implementation overhead needed to coordinate various states and makes the system unscalable. Typically, a core node is implemented with a limited number of logical buffers, where all the flows belonging to a given class are queued in a separate logical buffer served in a FIFO manner. We have seen that a FIFO system may cause a flow to hog the buffer resource and prevent other flows from accessing the buffer. An alternative approach to preventing this unfair buffer hogging behavior is to detect congestion when a buffer begins to reach a certain level and to appropriately notify sources to reduce the rate at which they send packets.

Random early detection (RED) is a buffer management technique that attempts to provide equitable access to a FIFO system by randomly dropping arriving packets before the buffer overflows. A dropped packet provides feedback information to the source (for example, via a missing acknowledgment) and informs the source to reduce its transmission rate. When a given source transmits at a higher rate than others, the source suffers from a higher packet-dropping rate. As a result, the given source reduces its transmission rate more, resulting in more uniform transmission rates among all the sources and more equitable access to the buffer resource.

The RED algorithm typically uses an average queue length rather than instantaneous queue length to decide how to drop packets. Specifically, two thresholds are defined: min_{th} and max_{th} . When the average queue length is below min_{th} , RED does not drop any arriving packets. When the average queue length is between min_{th} and max_{th} , RED drops an arriving packet with an increasing probability as the average queue length increases. This method of “early” drop is used to notify the source to reduce its transmission rate before the buffer becomes full. When the average queue length exceeds max_{th} , RED drops any arriving packet. Figure 7.49 shows an example of the packet drop probability as function of the average queue length.

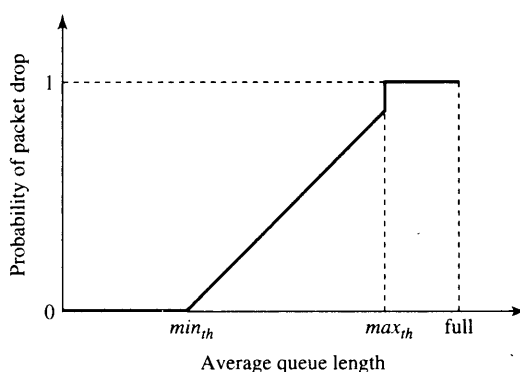


FIGURE 7.49 Packet drop profile in RED.

7.8 TRAFFIC MANAGEMENT AT THE FLOW LEVEL

At the flow level, traffic management is concerned with managing the individual traffic flow to ensure that the QoS (e.g., delay, jitter, loss) requested by the user is satisfied. Flow-level traffic management typically operates on the order of milliseconds to seconds. Packet switching has an advantage over circuit switching in terms of efficient resource utilization by allowing flows to dynamically share resources in the network. However, dynamic sharing may pose problems if the flow from each user is allowed to enter the network without some type of control. When too many packets contend for the same resource in the network, congestion occurs and network delay, loss, or throughput performance declines. For example, consider the packet-switching network shown in Figure 7.50. Suppose that nodes 1, 2, and 5 continuously transmit packets to their respective destinations via node 4. If the aggregate incoming rate of the packet flows to node 4 is greater than the rate the packets can be transmitted out, the buffer in node 4 will build up. If this situation persists, the buffer eventually becomes full and starts discarding packets. When the destination detects missing packets, it may ask the source to retransmit the packets. The source would unfortunately obey the protocol and send more packets to node 4, worsening the congestion even more. 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 traffic management at the flow level is to control the flows of traffic and maintain performance (controlled curve) even in the presence of traffic overload. The process of managing the traffic flow in order to control congestion is called **congestion control**.

It is tempting to claim that congestion can be solved by just allocating a large buffer. However, this solution merely delays the onset of congestion. 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 extremely large, packets will suffer from extremely long delays.

Congestion control is a very hard problem to solve, and many congestion control algorithms have been proposed. As with routing algorithms, we can classify congestion control algorithms several ways. The most logical approach identifies two broad classes: *open-loop control* and *closed-loop control*. Open-loop control prevents congestion from occurring by making sure that the traffic flow generated by the source will not degrade

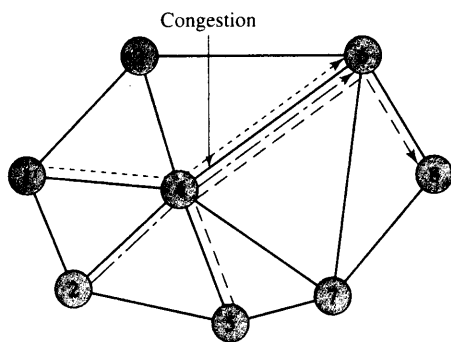


FIGURE 7.50 Congestion arises when incoming rate exceeds outgoing rate.

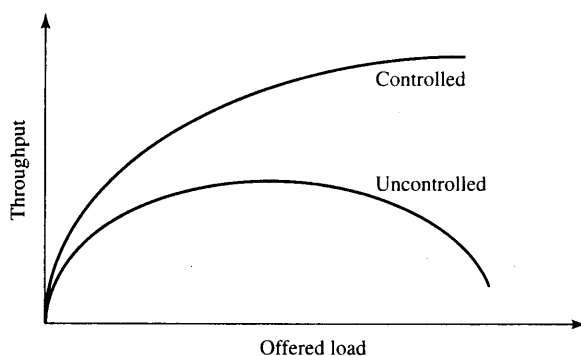


FIGURE 7.51 Throughput with and without congestion control.

the performance of the network to a level below the specified QoS. If the QoS cannot be guaranteed, the network rejects the traffic flow before it enters the network. The function that makes the decision to accept or reject a new traffic flow is called an *admission control*. Closed-loop control, on the other hand, reacts 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. Closed-loop control typically does not use any reservation.

7.8.1 Open-Loop Control

Open-loop control does not rely on feedback information to react to congestion. Instead, open-loop control is based on a principle that network performance is guaranteed to all traffic flows that have been admitted into the network. To guarantee network performance during the lifetime of admitted flows, open-loop control relies on three mechanisms: admission control, policing, and traffic shaping.

ADMISSION CONTROL

Admission control was initially developed for virtual-circuit packet-switching networks such as ATM but has been proposed for datagram networks as well. Admission control in ATM operates at the connection level and is therefore called connection admission control (CAC). Admission control in a datagram network makes sense only if packets of a given flow follow the same path.

The admission control entity is a network function that computes the resource (typically bandwidth and buffers) requirements of a new flow and determines whether the resources along the path to be followed by the flow are available. Thus a source initiating a new flow must first obtain permission from an admission control entity that decides whether the flow should be accepted or rejected. If the QoS of the new flow can be satisfied without violating QoS of existing flows, the flow is accepted; otherwise, the flow is rejected. The QoS may be expressed in terms of maximum delay, loss probability, delay variance, or other performance measures.

To determine whether the QoS of the flow can be satisfied, the admission control entity has to know the traffic parameters and the QoS requirements of the flow, which

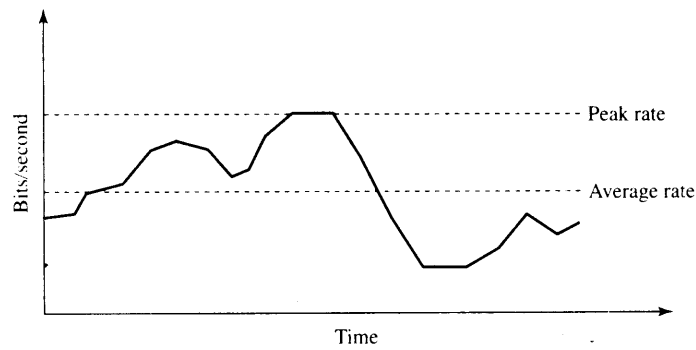


FIGURE 7.52 Example of a traffic flow.

in turn define the contract between the source of the flow and the network. The traffic parameters describe the traffic flow in a manner that can be readily quantified in the computation of QoS. Typical traffic parameters include peak rate (in bits/second or bytes/second), average rate (bits/second or bytes/second), and maximum burst size (in bits, bytes, or seconds). The peak rate defines the maximum rate that the source will generate its packets. The average rate defines the average rate that source will generate its packets. The maximum burst size determines the maximum length of time the traffic can be generated at the peak rate. Figure 7.52 shows an example of a traffic flow generated by a source, indicating the peak rate and the average rate. (Based on the traffic parameters and the QoS requirement, the admission control entity calculates how much bandwidth it has to reserve for the new flow.) The amount of bandwidth generally lies between the average rate and the peak rate and is called the *effective bandwidth* of the flow. The exact calculation for effective bandwidth is very complex and is beyond the scope of this book.

POLICING

Once a flow is accepted by an admission control entity, the QoS will be satisfied as long as the source obeys its negotiated traffic parameters during the lifetime of the flow. However, if the source violates the 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 continuously. The process of monitoring and enforcing the traffic flow is called **policing**. When the traffic flow violates the agreed-upon contract, the network may choose to discard or tag the nonconforming traffic. Tagging essentially lowers the priority of the nonconforming traffic, thus allowing the nonconforming traffic to be carried by the network as long as sufficient network resources are available. When network resources are exhausted, tagged traffic is the first to be discarded.

Most implementations of a policing device are based on the concept of a **leaky bucket**. Imagine the traffic flow to a policing device as water being poured into a bucket that has a hole at the bottom. The bucket has a certain depth and leaks at a constant rate when it is not empty. A new portion of water (that is, packet) is said to be *conforming* if the bucket does not overflow when the water is poured in the bucket.

When the bucket is full, the new portion of water is said to be *nonconforming* and the water can be discarded (or channeled to another bucket after being tagged). The size of the hole corresponds to the drain rate of the bucket. The hole ensures that the bucket will never overflow as long as the drain rate is higher than the rate water is being poured in. The bucket depth is used to absorb the fluctuations in the water flow. If we expect the water to flow into the bucket at a nearly constant rate, then the bucket can be made very shallow. If the water flow fluctuates widely (that is, there is bursty traffic), the bucket should be deeper.

There are many variations of the leaky bucket algorithm. In this section we look at an algorithm that is specified 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 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 7.53 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 typically depends on the traffic burstiness. At the arrival of the k th packet, the auxiliary variable X' records the difference between the bucket

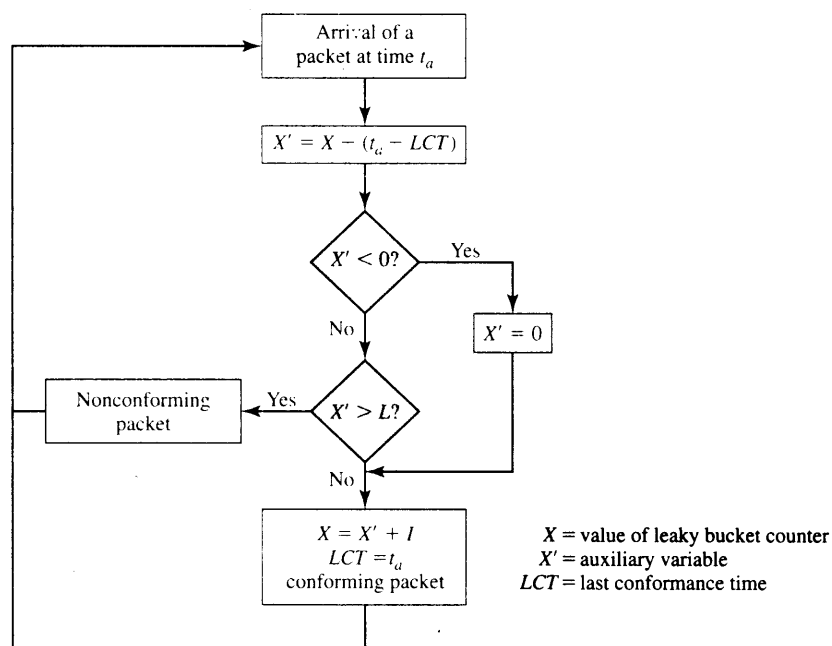


FIGURE 7.53 Leaky bucket algorithm used for policing.

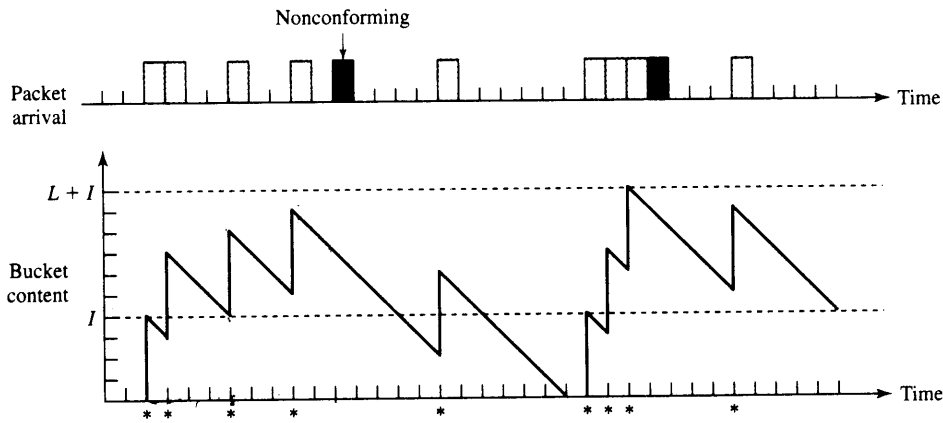


FIGURE 7.54 Behavior of leaky bucket.

content at the arrival of the last conforming packet and the interarrival time between the last conforming packet and the k th 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.

A simple example of the operation of the leaky bucket algorithm is illustrated in Figure 7.54. 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 $L + I$ (10). 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 number of packets that can be transmitted at the peak rate, called the **maximum burst size (MBS)**, is three. Note that the algorithm does not update the content of the bucket continuously but only at discrete points (arrival times of conforming packets) indicated by the asterisks. Also note that the values of I and L in general can take any real numbers.

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

$$MBS = 1 + \left\lceil \frac{L}{I - T} \right\rceil \quad (7.23)$$

where $\lceil x \rceil$ denotes 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 $L/(I - T)$ additional conforming packets. The relations among these quantities are pictorially depicted in Figure 7.55. MBS roughly

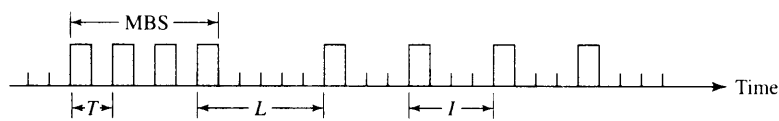


FIGURE 7.55 Relations among MBS and other parameters.

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.

A combination of leaky buckets can be used to police multiple traffic parameters (for example, the peak rate and the sustainable rate). In this situation *dual leaky buckets* such as the one shown in Figure 7.56 can be used. The traffic is first checked for the sustainable rate at the first leaky bucket. The nonconforming packets at the first bucket are dropped or tagged. The conforming (untagged) packets from the first bucket are then checked for the peak rate at the second bucket. The nonconforming packets at the second 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 exactly what its traffic flow looks like. If the source wants to ensure that the traffic flow conforms to the parameters specified in the leaky bucket policing device, it should first alter the traffic flow. **Traffic shaping** refers to the process of altering a traffic flow to ensure conformance. As shown in Figure 7.57, typically the traffic shaping device is located at the node just before the traffic flow leaves a network (egress node) while the policing device is located at the node that receives the traffic flow from another network (ingress node).

Traffic shaping can be realized in a number of ways. A *leaky bucket traffic shaper* is a very simple device, as shown in Figure 7.58. Incoming packets are first stored in a buffer. Packets (assumed to be of fixed length) are served periodically so that the

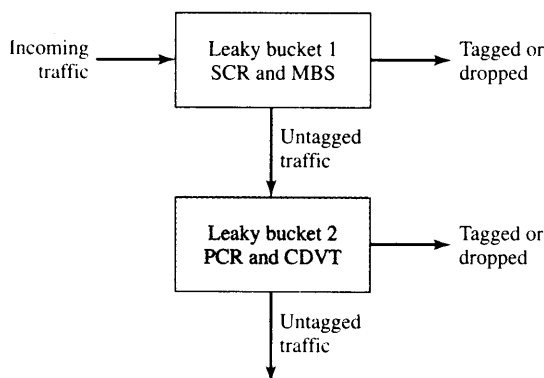


FIGURE 7.56 A dual leaky bucket configuration.

SCR = sustainable cell rate MBS = maximum burst size
 PCR = peak cell rate CDVT = cell delay variation tolerance

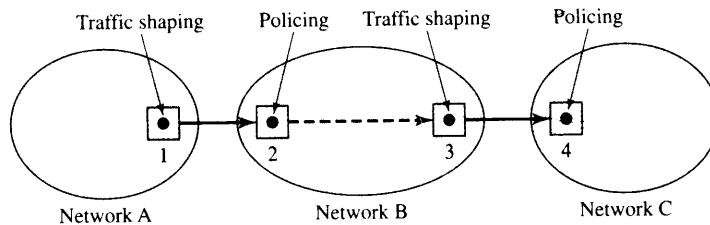


FIGURE 7.57 Typical locations of policing and traffic shaping devices.

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, and incoming packets are discarded when the buffer is full. A policing device checks and passes each packet on the fly. A traffic-shaping device needs to introduce certain delays for packets that arrive earlier than their scheduled departures and requires a buffer to store these packets.

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 the traffic flows from 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 device 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.

A more realistic 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 7.59, 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 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.

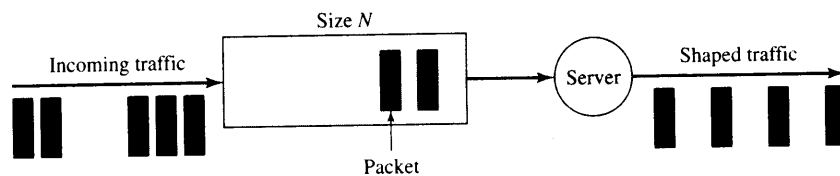


FIGURE 7.58 A leaky bucket traffic shaper.

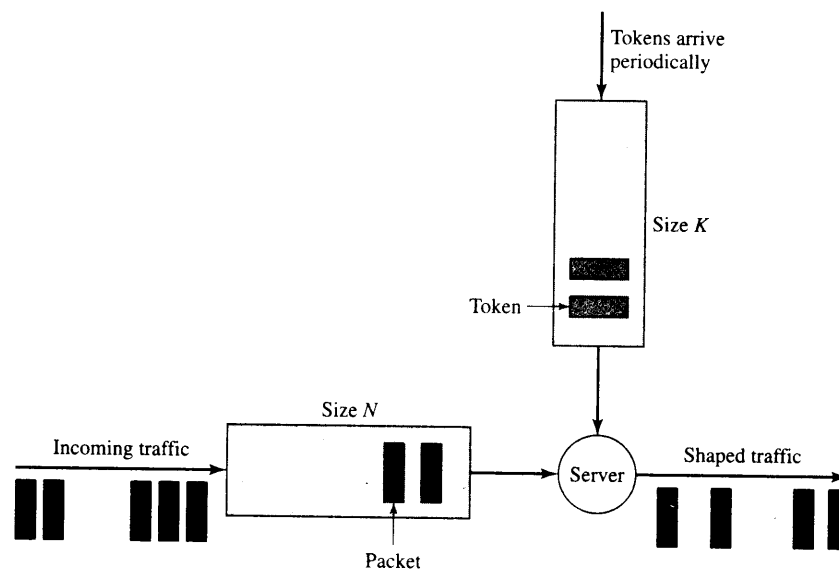


FIGURE 7.59 Token bucket traffic shaper.

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.

The token bucket traffic shaper can be implemented by an algorithm that is similar to the leaky bucket policing device shown in Figure 7.53. The corresponding token bucket algorithm for traffic shaping is left to the reader as an exercise.

QoS GUARANTEES AND SERVICE SCHEDULING

Switches and routers in packet-switching 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 7.60. Suppose we apply this traffic to two queueing systems in tandem each served by transmission lines of speed R bytes/second with $R > r$. We assume that the two queueing systems are empty and not serving any other flows.

Figure 7.61a shows the queueing system arrangement, and Figure 7.61b shows the buffer occupancy as a function of time. We assume that the token bucket allows an

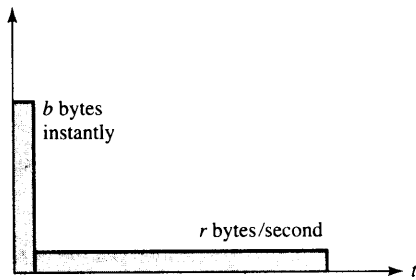


FIGURE 7.60 Maximum traffic allowed out of token bucket shaper.

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. An important observation is that the buffer occupancy at 1 is always less than b bytes. In addition we note that the buffer occupancy at a given time 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 queuing delay. We therefore conclude that the 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 queuing. 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]

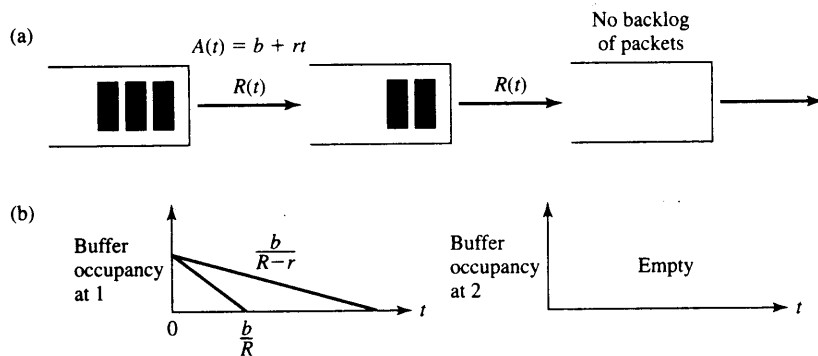


FIGURE 7.61 Delay experienced by token-bucket shaped traffic.

showed that if packet-by-packet weighted fair queuing 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:

$$D \leq \frac{b}{R} + \frac{(H-1)m}{R} + \sum_{j=1}^H \frac{M}{R_j} \quad (7.24)$$

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 procedure involves 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

The main objective of controlling congestion in a network is to maximize link utilization while preventing buffer overflows due to congestion. Maximizing link utilization calls for the sources to send packets as soon as packets are ready to be transmitted; preventing buffer overflows calls for the sources not to send too many packets. These two conflicting goals are the reason why congestion control is a delicate issue to tackle.

(Congestion control is usually addressed by a closed-loop control mechanism that relies on feedback information to regulate a packet flow rate according to feedback information about the state of the network, which may be based on buffer content, link utilization, or other relevant congestion information.) The recipient of the feedback information usually depends on the communication layer that is responsible for congestion control. In the TCP/IP environment, control is implemented at the transport layer, and thus the recipient of the feedback information usually resides at the source. In the ATM environment, control is implemented at the ATM layer corresponding to the network layer, and thus the recipient of the feedback information may reside at the intermediate node.

END-TO-END VERSUS HOP-BY-HOP

(With end-to-end closed-loop control, the feedback information about state of the network is propagated back to the source that can regulate the packet flow rate. The feedback information may be forwarded directly by a node that detects congestion, or it may be forwarded to the destination first which then relays the information to the source, as shown in Figure 7.62a. Because the transmission of the feedback information introduces a certain propagation delay, the information may not be accurate when the source receives such information.)

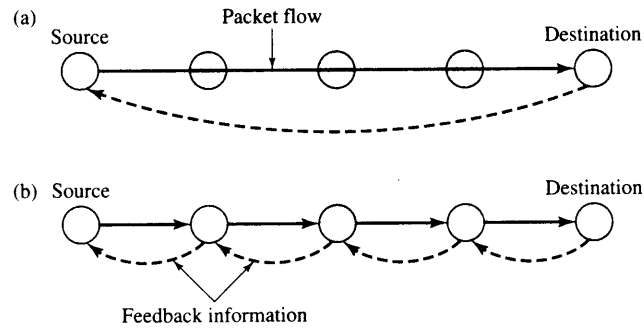


FIGURE 7.62 Closed-loop control: (a) end-to-end;
(b) hop-by-hop.

(Hop-by-hop control typically can react much faster than the end-to-end counterpart due to shorter propagation delay. With hop-by-hop closed-loop control, the state of the network is propagated to the upstream node, as shown in Figure 7.62b. When a node detects congestion on its outgoing link, it can tell its upstream neighbor to slow down its transmission rate. As a result, the upstream neighbor may also experience congestion some time later if the incoming rate exceeds the outgoing transmission rate. In turn this node tells its upstream neighbor to decrease the transmission rate. This “back-pressure” process from one downstream node to another node upstream may continue all the way to the source.)

IMPLICIT VERSUS EXPLICIT FEEDBACK

The feedback information can be implicit or explicit. With explicit feedback the node detecting congestion initiates an explicit message that eventually arrives at the source notifying congestion in the network. The explicit message can be transmitted as a separate packet (often called the *choke packet*), or piggybacked on a data packet. The contents of the explicit message can be in various forms. The simplest form is to use one bit of information (often called a binary feedback information) to indicate whether there is congestion or not. Another more elaborate form is to use richer information for the feedback information. One example is to transmit the message in the form of “desired rate” so that the recipient node may regulate its packet flow more precisely.

Closed-loop control in ATM networks is one example whereby each source continuously adjusts its sending rate according to explicit feedback information, which is recorded in a bit (called the EFCI bit) of the ATM cell header. When a node detects impending congestion, the node sets the EFCI bit of the data cells passing through the congestion link to 1. The destination that receives these cells with EFCI bit equal to 1 would send a special message to the corresponding source indicating that congestion has been detected and the source should throttle its transmission rate.

With implicit feedback, no such explicit message is forwarded. Instead, the source has to rely on some surrogate information to deduce congestion. One example is to use a time-out based on missing acknowledgments from a destination to decide whether congestion has been encountered in the network.

TCP congestion control is one example that regulates the transmission rate using the implicit feedback information derived from a missing acknowledgment, which triggers the retransmission timer to time out. When the source performs the time-out, it decreases the transmission rate by reducing its transmit window. The source then gradually increases the transmission rate until congestion is detected again, and the whole cycle repeats. TCP congestion control is discussed in more detail in Chapter 8.

7.9 TRAFFIC MANAGEMENT AT THE FLOW-AGGREGATE LEVEL

At the flow-aggregate level, traffic management deals with a multiplicity of flows. Flow-aggregate level works in a relatively long time scale on the order of minutes to days. Traffic management at the flow-aggregate level is often called **traffic engineering**. The main objective of traffic engineering is to map aggregated flows onto the network so that resources are efficiently utilized. We have seen that shortest-path routing allow traffic to be forwarded to a destination following the shortest path. Unfortunately, mapping the traffic according to shortest paths may not result in overall network efficiency, as shown in Figure 7.63a. In this example, traffic demands occur between sources 1, 2, and 3 and destination 8. With shortest path routing, the link connecting node 4 and node 8 is heavily utilized while other links are lightly loaded. Figure 7.63b shows an example of a better traffic mapping where the traffic is distributed across the network and the bottleneck link between node 4 and node 8 is removed.

The subject of traffic engineering is vast and complex. In general, effective traffic engineering relies on knowledge of the traffic demand information. In this section we discuss a simple technique that does not make use of traffic demand information and is called *constraint shortest-path routing*. The technique is suitable for connection-oriented packet-switching networks. Suppose that the traffic demand of bandwidth B between a given source and destination pair is to be routed. First, the algorithm prunes any link in the network that has available bandwidth less than B . Then, the algorithm runs the shortest path routing to find the paths between the given source and destination

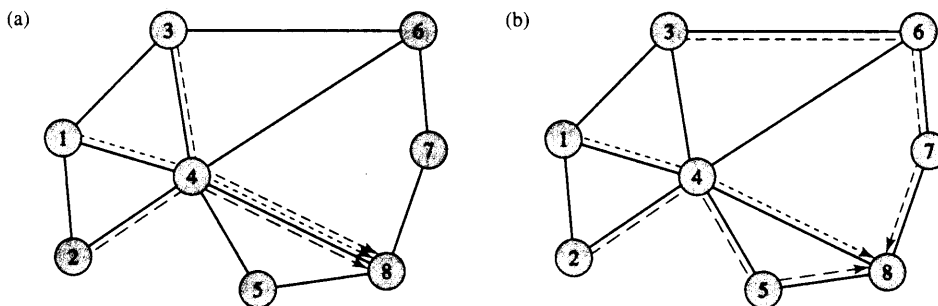


FIGURE 7.63 Mapping traffic onto the network topology.

pair. Consider the example in Figure 7.63 and suppose that we wish to set up three paths in the following order: node 1 to node 8 (path 1), node 2 to node 8 (path 2), and node 3 to node 8 (path 3). Assume that the bandwidth requested for each path is B and each link has a capacity of B . Initially, path 1 follows the shortest path $1 \rightarrow 4 \rightarrow 8$ using the original network topology. Link (1, 4) and link (4, 8) are then pruned. Next path 2 follows the shortest path $2 \rightarrow 4 \rightarrow 5 \rightarrow 8$ using the pruned network topology. Now links (2, 4), (4, 5), and (5, 8) are also pruned. Path 3 uses the revised pruned topology, which gives $3 \rightarrow 6 \rightarrow 7 \rightarrow 8$ as the shortest path.

Constraint shortest-path routing does not always yield a desired result. Consider now the case where the paths to be set up are given in the following order: path 2, path 1, and path 3. The reader can verify that path 2 follows $2 \rightarrow 4 \rightarrow 8$ and path 1 follows $1 \rightarrow 4 \rightarrow 6 \rightarrow 7 \rightarrow 8$. Path 3 cannot be successfully established in this case. This example shows that the order of path setup with constraint shortest-path routing plays an important role in the eventual path layout. An optimal path layout would have to consider all possible path requests at the same time so that optimization can be done globally.

SUMMARY

In this chapter we have examined packet-switching networks from several perspectives. We began by discussing the difference between the service offered by a network and the actual internal operation of the network. In particular we noted that the fact that a service is connection-oriented or connectionless does not imply that the internal operation uses the same mode.

We also examined packet-switching networks from a physical perspective and we traced the flow of packets from computers to LANs and routers in campus and wide area networks. We discussed the hierarchy of Internet service providers that is involved in the handling of Internet traffic.

We discussed two approaches to operating packet networks: virtual circuits and datagrams. The Internet and ATM networks were presented as examples. The advantages and disadvantages of the two approaches were discussed in terms of their complexity, their flexibility in dealing with failures, and their ability to provide quality of service. We also discussed the structure of packet switches.


We introduced several approaches to selecting routes across a network and we examined different types of routing tables that are involved in this process. We also discussed shortest-path algorithms and their use in synthesizing routing tables.

ATM networks were introduced as an example of connection-oriented networks. The rich set of techniques developed for ATM was used to introduce traffic management for packet networks in general. Traffic policing and shaping were introduced and their combination with scheduling in providing service guarantees was also discussed. Congestion control techniques for dealing with congestion in the network were introduced for IP and for ATM networks. Finally, the chapter concluded with an introduction to traffic engineering.

CHECKLIST OF IMPORTANT TERMS

asynchronous transfer mode (ATM)	packet
banyan switch	packet-switching network
Bellman-Ford algorithm	packet-by-packet fair queueing
cell	packet-by-packet weighted fair queueing
centralized routing	policing
congestion control	queue management
connection	queue scheduling
connectionless	random early detection (RED)
connection-oriented	routing
counting to infinity	shortest-path algorithm
cut-through packet switching	source routing
datagram packet switching	split horizon
deflection routing	split horizon with poisoned reverse
Dijkstra's algorithm	static routing
distance-vector protocol	store and forward
distributed routing	traffic engineering
dynamic (adaptive) routing	traffic management
explicit routing	traffic shaping
fair queueing	virtual channel identifier (VCI)
finish tag	virtual-channel connection (VCC)
first-in, first-out (FIFO) queueing	virtual circuit
flooding	virtual-circuit identifier (VCI)
head-of-line (HOL) blocking	virtual-circuit packet switching
head-of-line (HOL) priority queueing	virtual path
hop-by-hop routing	virtual path connection (VPC)
leaky bucket	virtual path identifier (VPI)
link-state protocol	weighted fair queueing
maximum burst size (MBS)	
message switching	

FURTHER READING

-  Bertsekas, D. and R. Gallager, *Data Networks*, Prentice-Hall, Englewood Cliffs, New Jersey, 1992.
- Floyd, S. and V. Jacobsen, "Random early detection gateways for congestion avoidance," *IEEE Transactions on Networking*, Vol. 1, No. 4, August 1993, pp. 397–413.
- 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.
- 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, Routers, Switches, and Internet Protocols*, Addison-Wesley, Reading, Massachusetts, 2000.
- 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.
- See our website for additional references available through the Internet.

PROBLEMS

- 7.1. 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?
- 7.2. Is it possible for a network to offer best-effort connection-oriented service? What features would such a service have, and how does it compare to best-effort connectionless service?
- 7.3. Suppose a service provider uses connectionless operation to run its network internally. Explain how the provider can offer customers reliable connection-oriented network service.
- 7.4. Where is complexity concentrated in a connection-oriented network? Where is it concentrated in a connectionless network?
- 7.5. Comment on the following argument: Because they are so numerous, end systems should be simple and dirt cheap. Complexity should reside inside the network.
- 7.6. 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.

- 7.7. Apply the end-to-end argument to the question of how to control the delay jitter that is incurred in traversing a multihop network.
- 7.8. Compare the operation of the layer 3 entities in the end systems and in the routers inside the network.
- 7.9. Consider a “fast” circuit-switching network in which the first packet in a sequence enters the network and triggers the setting up of a connection “on-the-fly” as it traces a path to its destination. Discuss the issues in implementing this approach in packet switching.
- 7.10. Circuit switching requires that the resources allocated to a connection be released when the connection has ended. Compare the following two approaches to releasing resources: (a) use an explicit connection release procedure where the network resources are released upon termination of the connection and (b) use a time-out mechanism where the resources allocated to a connection are released if a “connection refresh” message is not received within a certain time.
- 7.11. The oversubscription ratio can be computed with a simple formula by assuming that each subscriber independently transmits at the peak rate of c when there is data to send and zero otherwise. If there are N subscribers and the average transmission rate of each subscribers is r , then the probability of k subscribers transmitting data at the peak rate simultaneously is given by

$$P_k = \binom{N}{k} \left(\frac{r}{c}\right)^k \left(1 - \frac{r}{c}\right)^{N-k}$$

The access multiplexer is said to be in the “overflow mode” if there are more than n subscribers transmitting data simultaneously, and the corresponding overflow probability is given by

$$Q_n = \sum_{k=n+1}^N \binom{N}{k} \left(\frac{r}{c}\right)^k \left(1 - \frac{r}{c}\right)^{N-k}$$

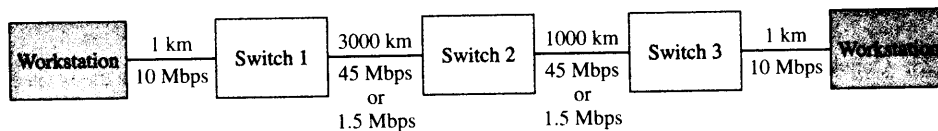
If a certain overflow probability is acceptable, then n is the smallest value that satisfy the given overflow probability. Find the oversubscription ratio if $N = 50$, $r/c = 0.01$, and $Q_n < 0.01$.

- 7.12. An access multiplexer serves $N = 1000$ subscribers, each of which has an activity factor $r/c = 0.1$. What is the oversubscription ratio if an overflow probability of 1 percent is acceptable? If $Np(1-p) \gg 1$, you may use an approximation technique (called the DeMoivre-Laplace Theorem), which is given by

$$P_k = \binom{N}{k} p^k (1-p)^{N-k} \approx \frac{1}{\sqrt{2\pi Np(1-p)}} e^{-\frac{(k-Np)^2}{2Np(1-p)}}$$

- 7.13. In Figure 7.5 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.
- Assume that the browser is on a computer that is in the same departmental LAN as the server.
 - Assume that the web server is in the central organization servers.
 - Assume that the server is located in a remote network.

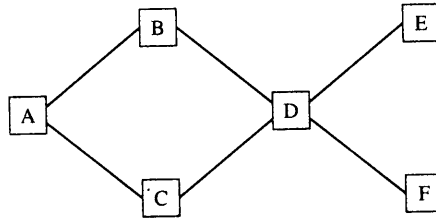
- 7.14. In Figure 7.5 trace the transmission of IP packets between two personal computers running an IP telephony application. Identify the components of the end-to-end delay.
- Assume that the two PCs are in the same departmental LAN.
 - Assume that the PCs are in different domains.
- 7.15. In Figure 7.5 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.
- 7.16. Explain why the distance in hops from your ISP to a NAP is very important. What happens if a NAP becomes congested?
- 7.17. A 64-kilobyte message is to be transmitted 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?
- 7.18. An audiovisual real-time application uses packet switching to transmit 32 kilobit/second speech and 64 kilobit/second video over the following network connection.



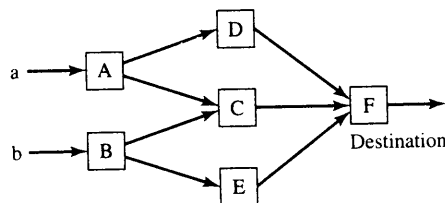
- Two choices of packet length are being considered: In option 1 a packet contains 10 milliseconds of speech and audio information; in option 2 a packet contains 100 milliseconds of speech and audio information. Each packet has a 40 byte header.
- For each option find out what percentage of each packet is header overhead.
 - Draw a time diagram and identify all the components of the end-to-end delay. 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 (that is, store and forward). Assume that bit errors are negligible.
 - 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.
 - Which of the preceding delay components would involve queueing delays?
- 7.19. 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.
- 7.20.** 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 transmission 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 (that is, the time between the first transmitted bit at the source and the last received bit at the destination)?
- (c) What value of k minimizes the total delay?
- 7.21.** Suppose that a datagram packet-switching 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?
- 7.22.** Suppose that a datagram packet-switching network uses headers of length H bytes and that a virtual-circuit packet-switching network uses headers of length h bytes. Use Figure 7.15 to determine the length M of a message for which the virtual-circuit network delivers the packet in less time than datagram network does. Assume packets in both networks are the same length.
- 7.23.** Consider the operation of a packet 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?
- 7.24.** Consider the operation of a packet 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?
- 7.25.** Consider the following traffic patterns in a banyan switch with eight inputs and eight outputs in Figure 7.21. Which traffic patterns below are successfully routed without contention? Can you give a general condition under which a banyan switch is said to be nonblocking (that is, performs successful routing without contention)?
- (a) Pattern 1: Packets from inputs 0, 1, 2, 3, and 4 are to be routed to outputs 2, 3, 4, 5, and 7, respectively.
- (b) Pattern 2: Packets from inputs 0, 1, 2, 3, and 4 are to be routed to outputs 1, 2, 4, 3, and 6, respectively.
- (c) Pattern 3: Packets from inputs 0, 2, 3, 4, and 6 are to be routed to outputs 2, 3, 4, 5, and 7, respectively.
- 7.26.** 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?
- 7.27.** Suppose that the virtual circuit identifiers (VCIs) are unique to a switch, not to an input port. What is traded off in this scenario?

- 7.28. Consider the virtual-circuit packet network in Figure 7.23. Suppose that node 4 in the network fails. Reroute the affected calls and show the new set of routing tables.
- 7.29. Consider the datagram packet network in Figure 7.25. Reconstruct the routing tables (using minimum-hop routing) that result after node 4 fails. Repeat if node 3 fails instead.
- 7.30. 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 network are modified to improve efficiency. Give the modified routing tables.
- 7.31. Consider the following six-node unidirectional network where flows a and b are to be transferred to the same destination. 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.
- 7.32. 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.
- 7.33. Consider the network in Figure 7.30.
- (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.

- 7.34.** Compare source routing with hop-by-hop routing with respect to (1) packet header overhead, (2) routing table size, (3) flexibility in route selection, and (4) QoS support, for both connectionless and connection-oriented packet networks.
- 7.35.** 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.
- 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 .
 - 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 .
- 7.36.** Consider a videoconferencing application in which the encoder produces a digital stream at a bit rate of 144 kbps. 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 7.35.
- Find an expression for the packetization delay for this video application as a function of P .
 - 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 .
- 7.37.** 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?
- 7.38.** Refer to the virtual-circuit packet-switching network in Figure 7.23. How many VCIs does each connection in the example consume? What is the effect of the length of routes on VCI consumption?
- 7.39.** Generalize the hierarchical network in Figure 7.26 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.
- How many nodes are in the hierarchy?
 - What does a routing table look like at level j in the hierarchy, $j = 1, 2, 3$, and 4 ?
 - What is the maximum number of hops between nodes in the network?
- 7.40.** 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 $1 \text{ cm} \times 1 \text{ cm}$ square on the surface of the earth?
- 7.41.** 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.
- What is the interval between production of full cells?
 - How long does it take to transmit the cell at 155 Mbps?
 - How many cells could be transmitted in this system between consecutive voice cells?
- 7.42.** 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?
- 7.43.** 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 $\alpha = 1$. Assume that the off period is also Pareto but with parameter α . If T has a Pareto distribution with parameter 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 produced?
- 7.44.** 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?
- 7.45.** 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.
- 7.46.** 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?
- 7.47.** Explain where priority queueing and fair queueing may be carried out in the generic switch/router in Figure 7.19.
- 7.48.** Consider the head-of-line priority system in Figure 7.42. 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.
- 7.49.** Consider the head-of-line priority system in Figure 7.42. 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.
- 7.50.** Incorporate some form of weighted fair queueing in the head-of-line priority system in Figure 7.42 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?
- 7.51.** Consider a packet-by-packet fair-queueing system with three logical buffers and with a service rate of one unit/second. Show the sequence of transmissions for this system for the

following packet arrival pattern. Buffer 1: arrival at time $t = 0$, length 2; arrival at $t = 4$, length 1. Buffer 2: arrival at time $t = 1$, length 3; arrival at $t = 2$, length 1. Buffer 3: arrival at time $t = 3$, length 5.

- 7.52.** Repeat Problem 7.51 if buffers 1, 2, and 3 have weights, 2, 3, and 5, respectively.
- 7.53.** 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.
- 7.54.** Deficit round-robin is a scheduling scheme that operates as follows. The scheduler visits the buffers in round-robin fashion. A deficit counter is maintained for each buffer. When the scheduler visits a buffer, 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 buffers 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 buffer as a function of time and indicate when the packets are transmitted.
- 7.55.** Should packet-level traffic management be performed in the core of the network where the packet streams have been aggregated to multiGigabit/second bit rates? Discuss the alternative of using TDM circuits to carry packet streams in the core network.
- 7.56.** Queue management with random early detection (RED):
- Explain why RED helps prevent TCP senders from detecting congestion and slowing down their transmission rates at the same time.
 - Discuss the effect of RED on network throughput.
 - Discuss the implementation complexity of the RED algorithm.
 - Discuss what would happen if instantaneous queue length were used instead of average queue length.
 - Explore ways to find reasonable values for the RED parameters (i.e., min_{th} , max_{th} , and the packet drop probability when the average queue length reaches max_{th}).
- 7.57.** 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 7.54, plot the bucket content and identify any nonconforming cells. Repeat if L is reduced to 4.
- 7.58.** Modify the leaky bucket algorithm in Figure 7.53 if packet length is variable.
- 7.59.** Explain the difference between the leaky bucket traffic shaper and the token bucket traffic shaper.
- 7.60.** Show the algorithm for token bucket traffic shaper using a flow chart similar to the one shown in Figure 7.53 for policing. The flow chart begins with a departure of a packet k at time $t_d(k)$ and calculates the departure time for packet $k + 1$. Define the necessary parameters and assume that packet length is fixed.
- 7.61.** Explain where the policing device and the traffic shaping device should be located in the generic packet switch in Figure 7.19.

- 7.62.** Which of the parameters in the upper bound for the end-to-end delay (Equation 7.24) are controllable by the application? What happens as the bit rate of the transmission links becomes very large?
- 7.63.** Suppose a source with an unlimited amount of information to transmit uses a closed-loop control to regulate the transmission rate according to feedback information. If the feedback information indicates that there is no congestion, the source continuously increases its transmission rate in a linear fashion. If the feedback information indicates congestion along the path, the source sets its transmission rate to zero and then repeats the cycle by continuously increasing its transmission rate until another congestion is detected. Assume that it takes T seconds for the feedback information to reach the source after congestion occurs. Sketch the transmission rate at the source versus time trajectory for a low and a high value of T . Explain how propagation delay T plays a role in closed-loop control.
- 7.64.** Consider the network in Figure 7.63. Suppose that paths need to be set up in the following order: nodes 5 to 8, 1 to 8, 2 to 4, 3 to 8, 3 to 5, 2 to 1, 1 to 3, 3 to 6, 6 to 7, and 7 to 8. Assume that each link has a capacity of one unit and each path requires bandwidth of one unit.
- Use shortest-path routing to set up as many paths as possible. How many paths are blocked?
 - Use constraint shortest-path routing to set up as many paths as possible. How many paths are blocked?
 - Can you suggest an improvement to constraint shortest-path routing?

CHAPTER 8

TCP/IP

The Internet Protocol (IP) enables communications across a vast and heterogeneous collection of networks that are based on different technologies. *Any* host computer that is connected to the Internet can communicate with any other computer that is also connected to the Internet. The Internet therefore offers ubiquitous connectivity and the economies of scale that result from large deployment.

The transport layer offers two basic communication services that operate on top of IP: Transmission Control Protocol (TCP) reliable stream service and User Datagram Protocol (UDP) datagram service. *Any* application layer protocol that operates on top of either TCP or UDP automatically operates across the Internet. Therefore the Internet provides a ubiquitous platform for the deployment of network-based services.

In Chapter 2 we introduced the TCP/IP protocol suite and showed how the various layers work together to provide end-to-end communications support for applications. In this chapter we examine the TCP/IP protocol suite in greater detail.

The chapter is organized as follows:

1. *TCP/IP architecture.* We have designed this book so that TCP/IP is introduced gradually throughout the text. In this section we summarize the TCP/IP concepts that are introduced in previous chapters. We also introduce commonly used definitions in the context of TCP/IP.
2. *Internet Protocol (IP).* We examine the structure of the network layer: the IP packet, the details of IP addressing, routing, and fragmentation and reassembly. We also discuss how IP is complemented by the Internet Control Message Protocol (ICMP).
3. *IP version 6.* We discuss the motivations for introducing a new version of IP, and we describe the features of IP version 6.
4. *Transport layer protocols.* We discuss the structure of the transport layer: UDP and TCP. We first examine UDP. We then examine in detail the structure of TCP and its protocol data unit (PDU), and we discuss the state transition diagram of the connection management process. We also describe the TCP congestion control mechanism.

5. *Internet routing.* A key element of IP is the routing protocols that are used to synthesize the routing tables that direct packets in the end systems and in the routers. We introduce the Routing Information Protocol (RIP) and the Open Shortest Path First (OSPF) Protocol for building routing tables within a domain. We also introduce the Border Gateway Protocol (BGP) for interdomain routing.
6. *Multicast routing.* We introduce approaches to multicast routing. We also discuss the Internet Group Management Protocol (IGMP) that enables hosts to join a multicast group.
7. *DHCP and mobile IP.* In the final section, we discuss two key protocols: Dynamic Host Configuration Protocol (DHCP) provides a mechanism for the temporary allocation of IP addresses to hosts; mobile IP allows a device to use the same IP address regardless of where it attaches to the network.

8.1 THE TCP/IP ARCHITECTURE

The TCP/IP protocol suite usually refers not only to the two most well-known protocols called the *Transmission Control Protocol (TCP)* and the *Internet Protocol (IP)* but also to other related protocols such as the *User Datagram Protocol (UDP)*, the *Internet Control Message Protocol (ICMP)* and the basic applications such as HTTP, TELNET, and FTP. The basic structure of the TCP/IP protocol suite is shown in Figure 8.1.

We saw in Chapter 2 that application layer protocols such as FTP and HTTP send messages using TCP. Application layer protocols such as SNMP and DNS send their messages using UDP. The PDUs exchanged by the peer TCP protocols are called **TCP segments** or **segments**, while those exchanged by UDP protocols are called **UDP datagrams** or **datagrams**. IP multiplexes TCP segments and UDP datagrams and performs fragmentation, if necessary, among other tasks to be discussed below. The

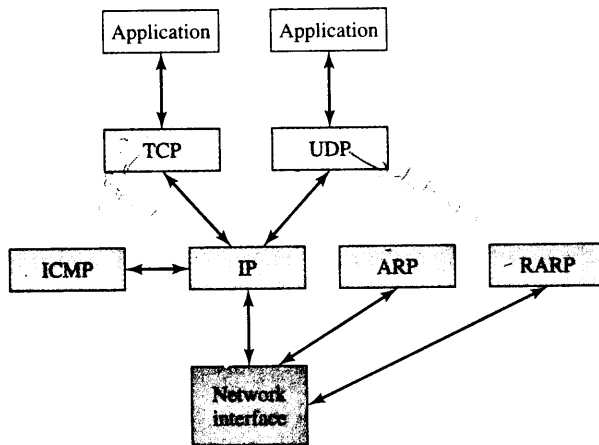


FIGURE 8.1 TCP/IP protocol suite.

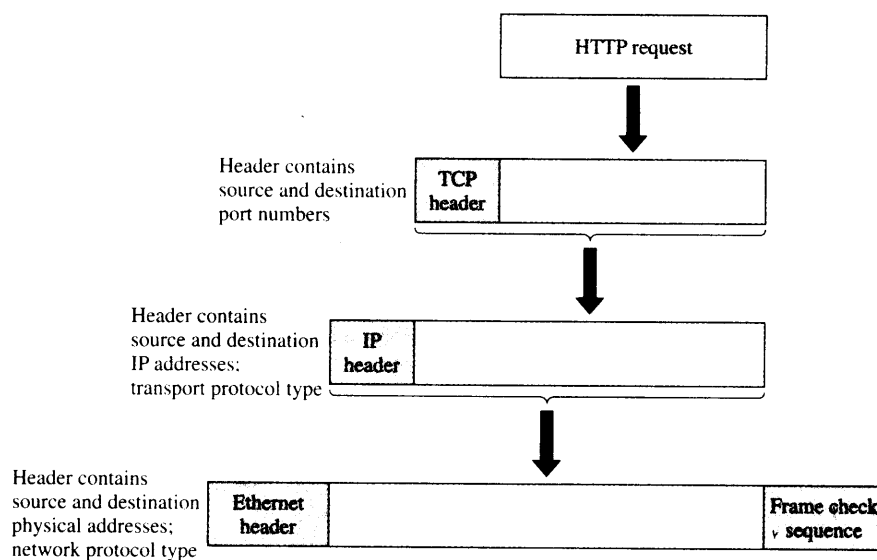


FIGURE 8.2 Encapsulation of PDUs in TCP/IP and addressing information in the headers (with HTTP as the application layer example).

protocol data units exchanged by IP protocols are called **IP packets** or **packets**.¹ IP packets are sent to the network interface for delivery across the physical network. At the receiver, packets passed up by the network interface are demultiplexed to the appropriate protocol (IP, ARP, or RARP). The receiving IP entity determines whether a packet should be sent to TCP or UDP. Finally, TCP (UDP) sends each segment (datagram) to the appropriate application based on the port number. The network interface can use a variety of technologies such as Ethernet, token ring, ATM, PPP over various transmission systems, and others.

The PDU of a given layer is encapsulated in a PDU of the layer below as shown in Figure 8.2. For example, an HTTP GET command is passed to the TCP layer, which encapsulates the message into a TCP segment. The segment header contains an ephemeral port number for the client process and the well-known port 80 for the HTTP server process. The TCP segment in turn is passed to the IP layer where it is encapsulated in an IP packet. The IP packet header contains an IP *network* address for the sender and an IP *network* address for the destination. IP network addresses are said to be *logical* because they are defined in terms of the *logical topology* of the routers and end systems. The IP packet is then passed through the network interface and encapsulated into a PDU of the underlying network. In Figure 8.2 the IP packet is encapsulated into an Ethernet frame. The frame header contains *physical* addresses that identify the physical endpoints for the Ethernet sender and the receiver. The logical IP

¹IP packets are sometimes called IP datagrams. To avoid confusion with UDP datagrams, in this text we use the term packets to refer to the PDUs at the IP layer.

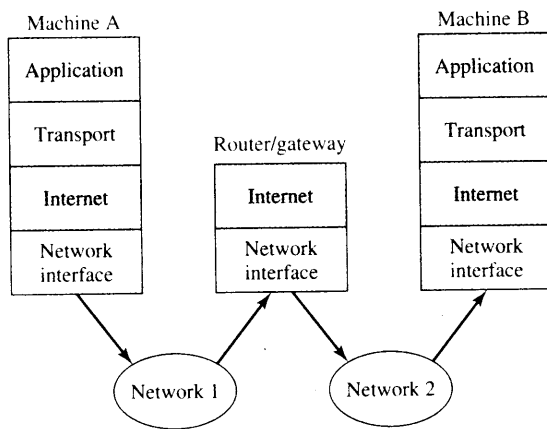


FIGURE 8.3 The Internet and network interface layers.

addresses need to be converted into specific physical addresses to carry out the transfer of bits from one device to the other. This conversion is done by an *address resolution protocol*.

Each host in the Internet is identified by a *globally unique IP address*. An IP address is divided into two parts: a *network ID* and a *host ID*. The network ID must be obtained from an organization authorized to issue IP addresses. The *Internet layer* provides for the transfer of information across multiple networks through the use of routers, as shown in Figure 8.3. IP packets are exchanged between routers without a connection setup; they are routed independently and may traverse different paths. The routers that interconnect the intermediate networks may discard packets when they encounter congestion. The responsibility for recovery from these losses is passed on to the transport layer.

The network interface layer is particularly concerned with the protocols that are used to access the intermediate networks. At each router the network interface layer is used to encapsulate the IP packet into a packet or frame of the underlying network or link. The IP packet is recovered at an exit router of the given network. This router must determine the next hop in the route to the destination and then encapsulate the IP packet into the frame of the type of the next network or link. This approach provides a clear separation of the Internet layer from the technology-dependent network interface layer. This approach also allows the Internet layer to provide a data transfer service that is transparent in the sense of not depending on the details of the underlying networks. Different network technologies impose different limits on the size of the blocks that they can handle. IP must accommodate the *maximum transmission unit* of an underlying network or link by implementing segmentation and reassembly as needed.

To enhance the scalability of the routing algorithms and to control the size of the routing tables, additional levels of hierarchy are introduced in the IP addresses. Within a domain the host address is further subdivided into a *subnetwork* part and an associated host part. This process facilitates routing within a domain, yet can remain transparent to the outside world. At the other extreme, addresses of multiple domains can be aggregated to create *supernets*. We discuss these key issues in Section 8.2.

8.2 THE INTERNET PROTOCOL

The Internet Protocol (IP) (RFC 791) is the heart of the TCP/IP protocol suite. IP corresponds to the network layer in the OSI reference model and provides a *connectionless best-effort delivery service* to the transport layer. Recall that a connectionless service does not require a virtual circuit to be established before data transfer can begin. The term *best-effort* indicates that IP will try its best to forward packets to the destination, but does not guarantee that a packet will be delivered to the destination. The term is also used to indicate that IP does not make any guarantee on the QoS.² An application requiring high reliability in packet delivery must implement the reliability function within a higher-layer protocol. Thus IP adopts the end-to-end argument discussed in Chapter 7 and relies on the end systems, if needed, to ensure that each packet transmitted at the source is received correctly at the destination. This design decision reduces the complexity of IP and increases its flexibility.

8.2.1 IP Packet

To understand the service provided by the IP entity, it is useful to examine the IP packet format, which contains a header part and a data part. The format of the IP header is shown in Figure 8.4.

The header has a fixed-length component of 20 bytes plus a variable-length component consisting of options that can be up to 40 bytes. IP packets are transmitted according to network byte order in the following groups: bits 0–7, bits 8–15, bits 16–23, and finally bits 24–31 for each row. The meaning of each field in the header follows.

Version: The version field indicates the version number used by the IP packet so that revisions can be distinguished from each other. The current IP version is 4. Version 5 is used for a real-time stream protocol called ST2, and version 6 is used for the new generation IP known as IPv6 (to be discussed in the Section 8.3).

0	4	8	16	19	24	31
Version	IHL	Type of service	Total length			
Identification			Flags	Fragment offset		
Time to live	Protocol		Header checksum			
Source IP address						
Destination IP address						
Options					Padding	

FIGURE 8.4 IP version 4 header.

²In Chapter 10 we discuss work on integrated services and differentiated IP services that provide some form of QoS in IP.

Internet header length: The Internet header length (IHL) specifies the length of the header in 32-bit words. If no options are present, IHL will have a value of 5. The length of the options field can be determined from IHL.

Type of service: The type of service (TOS) field traditionally specifies the priority of the packet based on delay, throughput, reliability, and cost requirements. Three bits are assigned for priority levels (called “precedence”) and four bits for the specific requirement (i.e., delay, throughput, reliability, and cost). For example, if a packet needs to be delivered to the destination as soon as possible, the transmitting IP module can set the delay bit to one and use a high-priority level. The TOS field is not in common use and so the field is usually set to zero. Recent work in the Differentiated Services Working Group of IETF redefines the TOS field in order to support other services that are better than the basic best effort. The differentiated services model is discussed in Chapter 10.

Total length: The total length specifies the number of bytes of the IP packet including header and data. With 16 bits assigned to this field, the maximum packet length is 65,535 bytes. In practice the maximum possible length is very rarely used, since most physical networks have their own length limitation. For example, Ethernet limits the payload length to 1500 bytes.

Identification, flags, and fragment offset: These fields are used for fragmentation and reassembly and are discussed below.

Time to live: The **time-to-live (TTL)** field is defined to indicate the amount of time in seconds the packet is allowed to remain in the network. However, most routers interpret this field to indicate the number of hops the packet is allowed to traverse in the network. Initially, the source host sets this field to some value. Each router along the path to the destination decrements this value by one. If the value reaches zero before the packet reaches the destination, the router discards the packet and sends an error message back to the source. With either interpretation, this field prevents packets from wandering aimlessly in the Internet.

Protocol: The protocol field specifies the upper-layer protocol that is to receive the IP data at the destination host. Examples of the protocols include TCP (protocol = 6), UDP (protocol = 17), and ICMP (protocol = 1).

Header checksum: The header checksum field verifies the integrity of the header of the IP packet. The data part is not verified and is left to upper-layer protocols. If the verification process fails, the packet is simply discarded. To compute the header checksum, the sender first sets the header checksum field to 0 and then applies the Internet checksum algorithm discussed in Chapter 3. Note that when a router decrements the TTL field, the router must also recompute the header checksum field.

Source IP address and destination IP address: These fields contain the addresses of the source and destination hosts. The format of the IP address is discussed below.

Options: The options field, which is of variable length, allows the packet to request special features such as security level, route to be taken by the packet, and timestamp at each router. For example, a source host can use the options field to specify a sequence of routers that a datagram is to traverse on its way to the destination host. Detailed descriptions of these options can be found in

[RFC 791]. Router alert is a new option introduced to alert routers to look inside the IP packet. The option is intended for new protocols that require relatively complex processing in routers along the path [RFC 2113]. It is used by RSVP, which is discussed in Chapter 10.

Padding: This field is used to make the header a multiple of 32-bit words.

When an IP packet is passed to the router, the following processing takes place. First the header checksum is computed and the fields in the header (version and total length) are checked to see if they contain valid values. Next the router identifies the next hop for the IP packet by consulting its routing table. Then the fields that need to be changed are updated. For example, the TTL and header checksum fields always require updating. The IP packet is then forwarded along the next hop.

Figure 8.5 shows an example of an IP packet header in an SMTP example. The middle pane shows all the fields of the IP packet header for frame 1. The packet uses IPv4 so the version number is 4. The header length is 20 bytes. The TOS field is shown as the Differentiated Service Field in keeping with recent redefinition of the field. The details of the flag field show that the “don’t fragment” bit is set. The TTL is set to 53 hops, and the protocol field is set to 6 for TCP. This is followed by the checksum and the source and destination IP addresses.

The screenshot shows the Wireshark interface for a capture named 'smtpex - Ethereal'. The main pane displays a list of 12 frames, all of which are SMTP commands and responses between source IP 66.185.95.99 and destination IP 192.168.2.18. The middle pane shows the expanded details of Frame 1 (205 bytes on wire, 205 bytes captured). The details pane is organized into sections: Ethernet II, Internet Protocol, and Simple Mail Transfer Protocol. The Internet Protocol section shows the following fields: Version: 4, Header Length: 20 bytes, Differentiated Services Field: 0x00 (DSCP 0x00: default; ECN: 0x00), Total Length: 191, Identification: 0x664a, Flags: 0x04 (Don't Fragment: Set), More Fragments: Not set, Fragment Offset: 0, Time to live: 53, Protocol: TCP (0x06), Header checksum: 0x7a18 (correct), Source: 66.185.95.99 (66.185.95.99), and Destination: 192.168.2.18 (192.168.2.18). The Simple Mail Transfer Protocol section shows: Command: MAIL FROM: [redacted]@rogers.com, Response: 250 Sender [redacted]@rogers.com ok, Command: RCPT TO: kate@accellight.com, Response: 250 Recipient kate@accellight.com ok, Command: DATA, Response: 354 ok Send data ending with <LF>.<LF>., and Command: Message Body.

FIGURE 8.5 Example of an IP packet header.

8.2.2 IP Addressing

To identify each node on the Internet, we have to assign a unique address to each node. A node (such as a router or a multihomed host) may have multiple network interfaces with each interface connected to a different network. An analogy of this situation is a house having multiple doors with one door facing a street called Main Street and another facing a different street called Broadway. In this situation an IP address is usually associated with the network interface or the network connection rather than with the node. Thus, in our analogy, an address is assigned to each door of a house rather than to the house itself. For a node with a single network interface (typically called a host), we can safely think of the IP address as the identity of the host.

An IP address has a fixed length of 32 bits. The address structure was originally defined to have a two-level hierarchy: network ID and host ID. The **network ID** identifies the network the host is connected to. Consequently, all hosts connected to the same network have the same network ID. The **host ID** identifies the network connection to the host rather than the actual host. An implication of this powerful aggregation concept is that a router can forward packets based on the network ID only, thereby shortening the size of the routing table significantly. The host ID is assigned by the network administrator at the local site. The network ID for an organization may be assigned by the ISP. An ISP in turn may request the network ID from its regional Internet registry: **American Registry for Internet Numbers (ARIN)**, **Réseaux IP Européens (RIPE)**, or **Asia Pacific Network Information Center (APNIC)**. When TCP/IP is used only within an intranet (an internal and private internet), the local network administrator may wish to assign the network ID on its own. However, the address will not be recognized by a host on the global Internet. The formats of the “classful” IP address are shown in Figure 8.6. The bit position shows the number of bits from the most significant bit.

The IP address structure is divided into five address classes: Class A, Class B, Class C, Class D, and Class E, identified by the most significant bits of the address as shown in the figure. Class A addresses have seven bits for network IDs and 24 bits

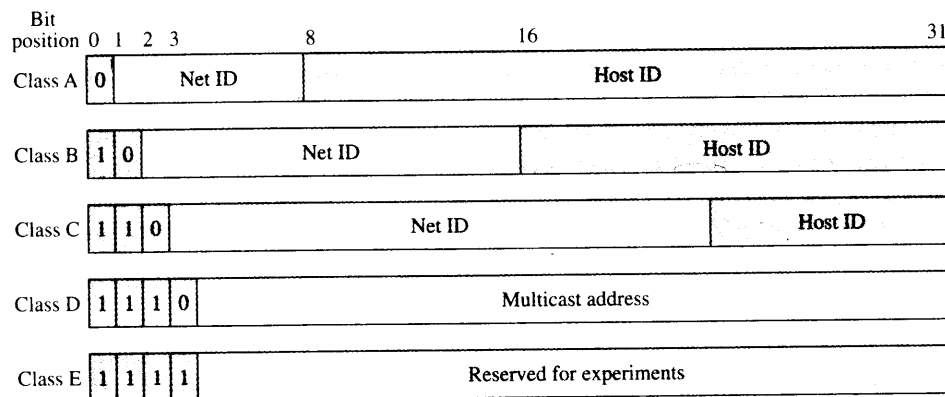


FIGURE 8.6 The five classes of IP addresses.

for host IDs, allowing up to 126 networks and about 16 million hosts per network. Class B addresses have 14 bits for network IDs and 16 bits for host IDs, allowing about 16,000 networks and about 64,000 hosts for each network. Class C addresses have 21 bits for network IDs and 8 bits for host IDs, allowing about 2 million networks and 254 hosts per network. Class D addresses are used for multicast services that allow a host to send information to a group of hosts simultaneously. Class E addresses are reserved for experiments.

An ID that contains all 1s or all 0s has a special purpose. A host ID that contains all 1s is meant to broadcast the packet to all hosts on the network specified by the network ID. If the network ID also contains all 1s, the packet is broadcast on the local network. A host ID that contains all 0s refers to the network specified by the network ID, rather than to a host. It is possible for a host not to know its IP address immediately after being booted up. In this case the host may transmit packets with all 0s in the source address in an attempt to find out its own IP address. The machine is identified by its MAC address. Other hosts interpret the packet as originating from “this” host.

IP addresses are usually written in **dotted-decimal notation** so that they can be communicated conveniently by people. The address is broken into four bytes with each byte being represented by a decimal number and separated by a dot. For example, an IP address of

10000000 10000111 01000100 00000101

is written as

128.135.68.5

in dotted-decimal notation. The discerning student should notice immediately that this address is a Class B address. As we saw before, some of the values of the address fields (such as all 0s and all 1s) are reserved for special purposes. Another important special value is 127.X.Y.Z (X, Y, and Z can be anything), which is used for loopback. When a host sends a packet with this address, the packet is returned to the host by the IP protocol software without transmitting it to the physical network. The loopback address can be used for interprocess communication on a local host via TCP/IP protocols and for debugging purposes.

A set of specific ranges of IP addresses have been set aside for use in private networks (RFC 1918). These addresses are used within internets that do not connect directly to the Internet, for example, home networks. These addresses are considered *unregistered* and routers in the Internet must discard packets with these addresses. A range of addresses has been defined for each IP class:

Range 1:	10.0.0.0	to	10.255.255.255
Range 2:	172.16.0.0	to	172.31.255.255
Range 3:	192.168.0.0	to	192.168.255.255 (used in home LANs)

Network address translation is used to connect to the global Internet.

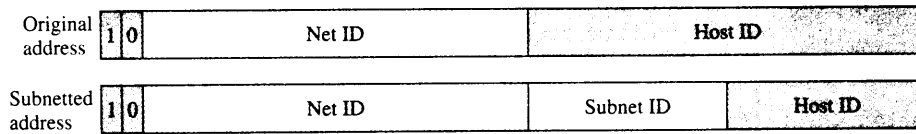


FIGURE 8.7 Introducing another hierarchical level through subnet addressing.

8.2.3 Subnet Addressing

The original IP addressing scheme described above has some drawbacks. Consider a typical university in the United States that has a Class B network address (which can support about 64,000 hosts connected to the Internet). With the original addressing scheme, it would be a gigantic task for the local network administrator to manage all 64,000 hosts. Moreover, a typical campus would have more than one local network, requiring the use of multiple network addresses. To solve these problems, subnet addressing was introduced in the mid-1980s when most large organizations began moving their computing platforms from mainframes to networks of workstations. The basic idea of **subnetting** is to add another hierarchical level called the “subnet” as shown in Figure 8.7. The beauty of the subnet-addressing scheme is that it is oblivious to the network outside the organization. That is, a host outside this organization would still see the original address structure with two levels. Inside the organization the local network administrator is free to choose any combination of lengths for the subnet and host ID fields.

As an illustration, consider an organization that has been assigned a Class B IP address with a network ID of 150.100. Suppose the organization has many LANs, each consisting of no more than 100 hosts. Then seven bits are sufficient to uniquely identify each host in a subnetwork. The other nine bits can be used to identify the subnetworks within the organization. If a packet with a destination IP address of 150.100.12.176 arrives at the site from the outside network, which subnet should a router forward this packet to? To find the subnet number, the router needs to store an additional quantity called a **subnet mask**, which consists of binary 1s for every bit position of the address except in the host ID field where binary 0s are used. For our example, the subnet mask is

11111111 11111111 11111111 10000000

which corresponds to 255.255.255.128 in dotted-decimal notation. The router can determine the subnet number by performing a binary AND between the subnet mask and the IP address. In our example the IP address is given by

10010110 01100100 00001100 10110000

Thus the subnet number becomes

10010110 01100100 00001100 10000000

corresponding to 150.100.12.128 in dotted-decimal notation. This number is used to forward the packet to the correct subnetwork inside the organization. Note that IP address 150.100.12.128 is used to identify the subnetwork and IP address 150.100.12.255 is used to broadcast packets inside the subnetwork. Thus a host connected to this subnetwork must have an IP address in the range 150.100.12.129 to 150.100.12.254.

EXAMPLE Subnetwork Addressing

Consider a site that has been assigned a Class B IP address of 150.100.0.1, as shown in Figure 8.8. The site has a number of subnets and many hosts (H) connected by routers (R). The figure shows only three subnets and five hosts for simplicity. Assume that the subnet ID field is nine bits long and the host ID field is seven bits long.

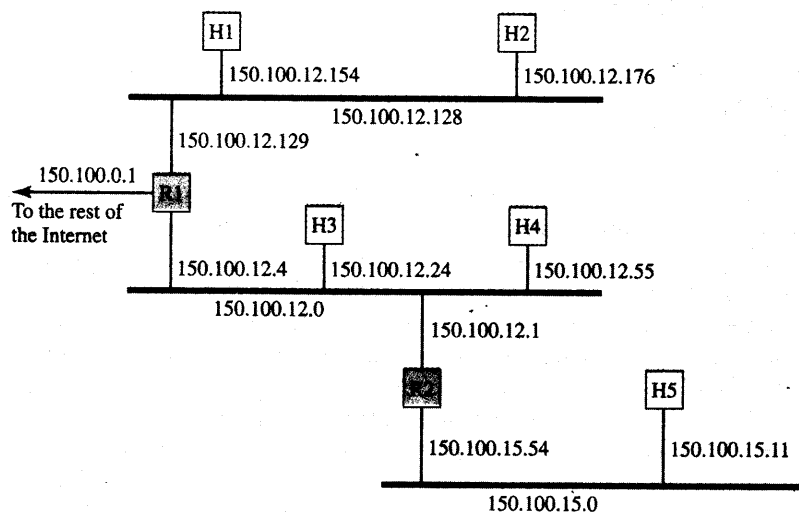


FIGURE 8.8 Example of address assignment with subnetting.

When a host located outside this network wants to send a packet to a host on this network, all that external routers have to know is how to get to network address 150.100.0.1. This concept is very powerful, since it hides the details of the internal network configuration. Let's see how the internal routers handle arriving packets.

Suppose a packet having a destination IP address of 150.100.15.11 arrives from the outside network. R1 has to know the next-hop router to send the packet to. The address 150.100.15.11 corresponds to the binary string 10010110 01100100 00001111 00001011.³ R1 knows that a nine-bit subnet field is in use, so it applies the following mask to extract the subnetwork address from the IP address: 11111111 11111111 11111111 10000000. The result is then 10010110 01100100 00001111 00000000, which corresponds to 150.100.15.0. Router R1 looks up this subnet number in its routing table, and finds the corresponding entry to specify the next-hop router address for R2, which is 150.100.12.1. When R2 receives the packet, R2 performs the same process and finds out that the destination host is connected to one of its network interfaces. It can thus send the packet directly to the destination.

³150 = 128 + 16 + 4 + 2, which gives 10010110; similarly 100 = 64 + 32 + 4; 15 = 8 + 4 + 2 + 1; 11 = 8 + 2 + 1.

8.2.4 IP Routing

The IP layer in the end-system hosts and in the routers work together to route packets from IP network sources to destinations. The IP layer in each host and router maintains a routing table that it uses to determine how to handle each IP packet. Consider the action of the originating host. If its routing table indicates that the destination host is directly connected to the originating host by a link or by a LAN, then the IP packet is sent directly to the destination host using the appropriate network interface. Otherwise, the routing table typically specifies that the packet is to be sent to a default router that is directly connected to the originating host. Now consider the action of a router. When a router receives an IP packet from one of the network interfaces, the router examines its routing table to see whether the packet is destined to itself, and if so, delivers the packet to the appropriate higher-layer protocol. If the destination IP address is not the router's own address, then the router determines the next-hop router and the associated network interface, and then forwards the packet.

Each row in the routing table must provide the following information: destination IP address; IP address of next-hop router; several flag fields; outgoing network interface; and other information such as subnet mask, physical address, and statistics information. Several types of flags may be defined. For example, the H flag indicates whether the route in the given row is to a host ($H = 1$) or to a network ($H = 0$). The G flag indicates whether the route in the given row is to a router (gateway; $G = 1$) or to a directly connected destination ($G = 0$).

Each time a packet is to be routed, the routing table is searched in the following order. First, the destination column is searched to see whether the table contains an entry for the complete destination IP address. If so, then the IP packet is forwarded according to the next-hop entry and the G flag. Second, if the table does not contain the complete destination IP address, then the routing table is searched for the destination network ID. If an entry is found, the IP packet is forwarded according to the next-hop entry and the G flag. Third, if the table does not contain the destination network ID, the table is searched for a default router entry, and if one is available, the packet is forwarded there. Finally, if none of the above searches are successful, then the packet is declared undeliverable and an ICMP "host unreachable error" packet is sent back to the originating host. (ICMP is discussed in Section 8.2.9.)

EXAMPLE Routing with Subnetworks

Suppose that host H5 wishes to send an IP packet to host H2 in Figure 8.8. H2 has IP address 150.100.12.176 (10010110 01100100 00001100 10110110). Let us trace the operations in carrying out this task.

The routing table in H5 may look something like this:

Destination	Next-Hop	Flags	Network Interface
127.0.0.1	127.0.0.1	H	lo0
default	150.100.15.54	G	emd0
150.100.15.0	150.100.15.11		emd0

The first entry is the loopback interface where the H indicates a host address, and 100 by convention is always the loopback interface. The second entry is the default entry, with next-hop router R2 (150.100.15.54), which is a router, so G = 1, and with Ethernet interface emd0. The third entry does not have H set, so it is a network address; G is also not set, so a direct route is indicated and, by convention, the next-hop entry is the IP address of the outgoing network interface.

H5 first searches its routing table for the IP packet destination address 150.100.12.176. When H5 does not find the entry, it then searches for the destination network ID 150.100.12.128. Finally, H5 finds the default route to R2 and forwards the IP packet across the Ethernet.

The routing table in R2 may look something like this:

Destination	Next-Hop	Flag	Network Interface
127.0.0.1	127.0.0.1	H	lo0
default	150.100.12.4	G	emd0
150.100.15.0	150.100.15.54		emd1
150.100.12.0	150.100.12.1		emd0

R2 searches its routing table and forwards the IP packet to router R1, using the default route. R1 has the following entries in its routing table:

Destination	Next-Hop	Flag	Network Interface
127.0.0.1	127.0.0.1	H	lo0
150.100.12.176	150.100.12.176		emd0
150.100.12.0	150.100.12.4		emd1
150.100.15.0	150.100.12.1	G	emd1

R1 searches its routing table and finds a match to the host address 150.100.12.176 and sends the packet through network interface emd0, which delivers the packet to H2.

The `netstat` command allows you to display the routing table in your workstation. Check the manual for your system on how to use this command.

8.2.5 Classless Interdomain Routing (CIDR)

Dividing the IP address space into A, B, and C classes turned out to be inflexible. While on the one hand most organizations utilize the Class B address space inefficiently, on the other hand most organizations typically need more addresses than can be provided by a Class C address space. Giving a Class B address space to each organization would have exhausted the IP address space easily because of the rapid growth of the Internet. In 1993 the classful address space restriction was lifted. An arbitrary prefix length to indicate the network number, known as **classless interdomain routing (CIDR)**,⁴ was adopted in place of the classful scheme [RFC 1518]. Using a CIDR notation, a prefix 205.100.0.0 of length 22 is written as 205.100.0.0/22. The /22 notation indicates that the network mask is 22 bits, or 255.255.252.0.

⁴CIDR is pronounced like "cider."

With CIDR, packets are routed according to the prefix of the IP address without distinguishing different address classes. The entries in a CIDR routing table contain a 32-bit IP address and a 32-bit mask. CIDR enables a technique called **supernetting** to allow a single routing entry to cover a block of classful addresses. For example, instead of having four entries for a *contiguous* set of Class C addresses (e.g., 205.100.0.0, 205.100.1.0, 205.100.2.0, and 205.100.3.0), CIDR allows a single routing entry 205.100.0.0/22, which includes all IP addresses from 205.100.0.0 to 205.100.3.255. To see the route aggregation process in more detail, we note that the original four Class C entries

```
Class C address 205.100.0.0 = 11001101 01100100 00000000 00000000
Class C address 205.100.1.0 = 11001101 01100100 00000001 00000000
Class C address 205.100.2.0 = 11001101 01100100 00000010 00000000
Class C address 205.100.3.0 = 11001101 01100100 00000011 00000000
```

become

```
Mask           255.255.252.0 = 11111111 11111111 11111100 00000000
Supernet address 205.100.0.0 = 11001101 01100100 00000000 00000000
```

RFC 1518 describes address allocation policies to capitalize on CIDR's ability to aggregate routes. For example, address assignments should reflect the physical topology of the network; in this case IP address prefixes should correspond to continents or nations. This approach facilitates the aggregation of logical packet flows into the physical flows that ultimately traverse the network. Similarly, *transit routing domains* that carry traffic between domains should have unique IP addresses, and domains that are attached to them should begin with the transit routing domain's prefix. These route aggregation techniques resulted in a significant reduction in routing table growth, which was observed after the deployment of CIDR. Without the CIDR deployment, the default-free routing table size at the core of the Internet would have easily exceeded 100,000 routes in 1996. CIDR was able to reduce the routing table size to around 50,000 routes in 1998, and in 2003 the table size has grown to slightly above 100,000 routes.

The use of variable-length prefixes requires that the routing tables be searched to find the **longest prefix match**. For example, a routing table may contain entries for the above supernet 205.100.0.0/22 as well as for the even larger supernet 205.100.0.0/20. This situation may arise when a large number of destinations have been aggregated into the block 205.100.0.0/20, but packets destined to 205.100.0.0/22 are to be routed differently. A packet with destination address 205.100.1.1 will match both of these entries, so the algorithm must select the match with the longest prefix.

Routing tables can contain tens of thousands of entries, so efficient, fast, longest-prefix matching algorithms are essential to implement fast routers. A number of algorithms have been developed to perform table lookup. For example, see [Degermark 1998] and [Waldvogel 1998].

8.2.6 Address Resolution

In Section 8.2.4, we assume that a host can send a packet to the destination host by knowing the destination IP address. In reality IP packets must eventually be delivered by

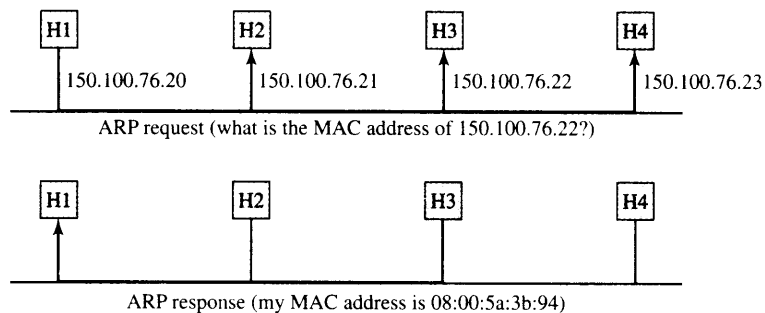


FIGURE 8.9 Address Resolution Protocol.

the underlying network technology, which uses its own addressing scheme. Currently, Ethernet is the most common underlying network technology that IP runs on. Recall that the Ethernet hardware can understand only its own 48-bit MAC address format. Thus the source host must also know the destination MAC address if the packet is to be delivered to the destination host successfully.

How does the host map the IP address to the MAC address? An elegant solution to find the destination MAC address is to use the **Address Resolution Protocol (ARP)**. The main idea is illustrated in Figure 8.9. Suppose H1 wants to send an IP packet to H3 but does not know the MAC address of H3. H1 first broadcasts an ARP request packet asking the destination host, which is identified by H3's IP address, to reply. All hosts in the network receive the packet, but only the intended host, which is H3, responds to H1. The ARP response packet contains H3's MAC and IP addresses. From now on H1 knows how to send packets to H3. To avoid having to send an ARP request packet each time H1 wants to send a packet to H3, H1 caches H3's IP and MAC addresses in its ARP table so that H1 can simply look up H3's MAC address in the table for future use. Each entry in the ARP table is usually "aged" so that the contents are erased if no activity occurs within a certain period, typically around 5 to 30 minutes. This procedure allows changes in the host's MAC address to be updated. The MAC address may change, for example, when an Ethernet card is broken and is replaced with a new one.

Figure 8.10 uses a sequence of POP3 messages to provide an example of an ARP request from the source with MAC address 00:01:03:1d:cc:f7, which is manufactured by 3COM. The first frame in the top pane shows that the ARP request is for the MAC address of the machine with IP address 192.168.2.18. The middle pane provides additional details for the first frame. It can be seen that that frame is an Ethernet frame that has the destination address set for broadcast and the type field set to ARP (0x0806). The ARP packet indicates that the hardware is Ethernet, that the message is an ARP request, and that the target is a 4-byte IP address. This is followed by the sender's IP (192.168.2.18) and MAC address. The target IP address (192.168.2.1) is shown and the target MAC address contains all zeros indicating that it is not known. The second frame in the top pane contains the reply from the target machine indicating that its MAC address is 00:04:e2:29:b2:3a, which is manufactured by SMC.

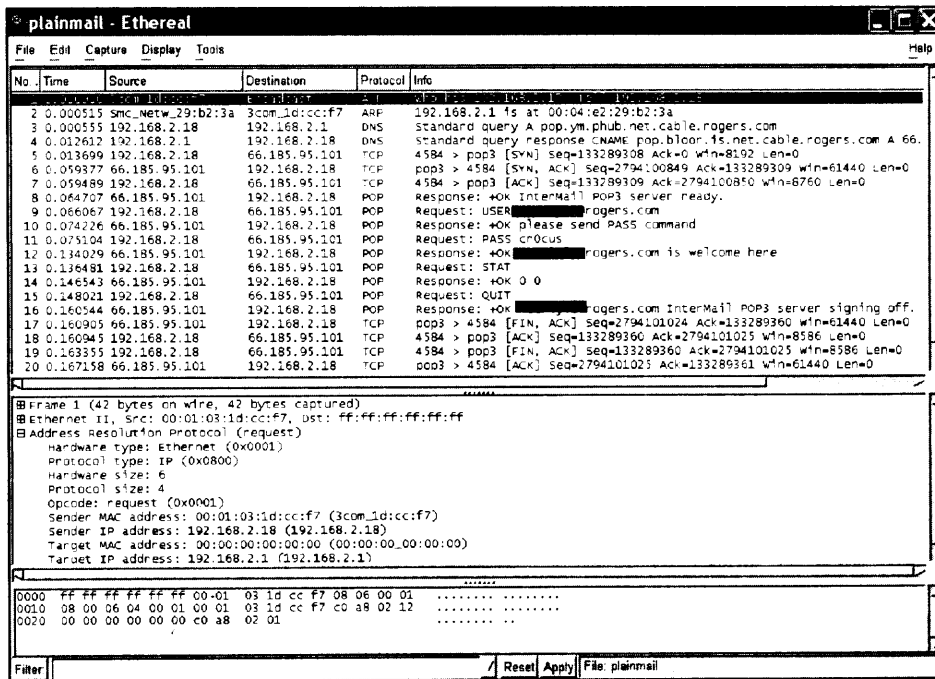


FIGURE 8.10 Example of an ARP request.

8.2.7 Reverse Address Resolution

In some situations a host may know its MAC address but not its IP address. For example, when a diskless computer such as an X terminal is being bootstrapped, it can read the MAC address from its Ethernet card. However, its IP address is usually kept separately in a disk at the server. The problem of getting an IP address from a MAC address can be handled by the **Reverse Address Resolution Protocol (RARP)**, which works in a fashion similar to ARP.

To obtain its IP address, the host first broadcasts an RARP request packet containing its MAC address on the network. All hosts on the network receive the packet, but only the server replies to the host by sending an RARP response packet containing the host's MAC and IP addresses. One limitation with RARP is that the server must be located on the same physical network as the host.

8.2.8 Fragmentation and Reassembly

One of the strengths of IP is that it can work on a variety of physical networks. Each physical network usually imposes a certain packet-size limitation on the packets that can be carried, called the **maximum transmission unit (MTU)**. For example, Ethernet specifies an MTU of 1500 bytes, and FDDI specifies an MTU of 4464 bytes. When IP

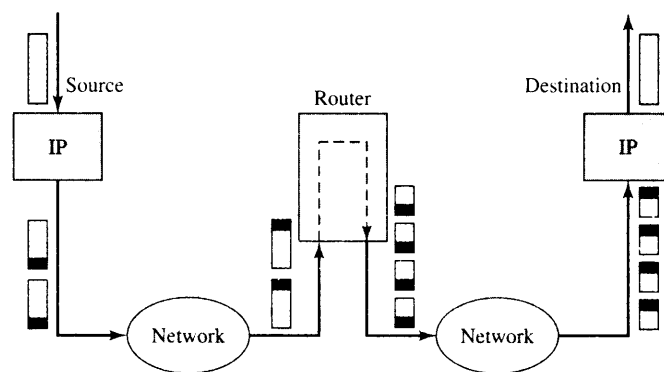


FIGURE 8.11 Packet fragmentation.

has to send a packet that is larger than the MTU of the physical network, IP must break the packet into smaller **fragments** whose size can be no larger than the MTU. Each fragment is sent independently to the destination as though it were an IP packet. If the MTU of some other network downstream is found to be smaller than the fragment size, the fragment will be broken again into smaller fragments, as shown in Figure 8.11. The destination IP is the only entity that is responsible for reassembling the fragments into the original packet. To reassemble the fragments, the destination waits until it has received all the fragments belonging to the same packet. If one or more fragments are lost in the network, the destination abandons the reassembly process and discards the rest of the fragments. To detect lost fragments, the destination host sets a timer once the first fragment of a packet arrives. If the timer expires before all fragments have been received, the host assumes the missing fragments were lost in the network and discards the other fragments.

Three fields in the IP header (identification, flags, and fragment offset in Figure 8.4) have been assigned to manage fragmentation and reassembly. At the destination IP has to collect fragments for reassembling into packets. The identification field is used to identify which packet a particular fragment belongs to so that fragments for different packets do not get mixed up. To have a safe operation, the source host must not repeat the identification value of the packet destined to the same host until a sufficiently long period of time has passed.

The flags field has three bits: one unused bit, one “don’t fragment” (DF) bit, and one “more fragment” (MF) bit. If the DF bit is set to 1, it forces the router not to fragment the packet. If the packet length is greater than the MTU, the router will have to discard the packet and send an error message to the source host. The MF bit tells the destination host whether or not more fragments follow. If there are more, the MF bit is set to 1; otherwise, it is set to 0. The fragment offset field identifies the location of a fragment in a packet. The value measures the offset, in units of eight bytes, between the beginning of the packet to be fragmented and the beginning of the fragment, considering the data part only. Thus the first fragment of a packet has an offset value of 0. The data length of each fragment, except the last one, must be a multiple of eight bytes. The reason the offset is measured on units of eight bytes is that the fragment offset field has only

13 bits, giving a maximum count of 8192. To be able to cover all possible data lengths up to the maximum of 65,536 bytes, it is sufficient to multiply 8192 by 8. The reader should verify that these three fields give sufficient information to hosts and routers to perform fragmentation and reassembly.

Although fragmentation may seem to be a good feature to implement, it involves a subtle performance penalty. As alluded to earlier, if one of the fragments is lost, the packet cannot be reassembled at the destination and the rest of the fragments have to be discarded. This process, of course, wastes transmission bandwidth. If the higher-layer protocol requires reliability, then all fragments for that packet would have to be retransmitted. It is possible to save some bandwidth if routers discard fragments more intelligently. Specifically, if a router has to discard a fragment, say, due to congestion, it might as well discard the subsequent fragments belonging to the same packet, since they will become useless at the destination. In fact, this same idea has been implemented in ATM networks.

EXAMPLE Fragmenting a Packet

Suppose a packet arrives at a router and is to be forwarded to an X.25 network having an MTU of 576 bytes. The packet has an IP header of 20 bytes and a data part of 1484 bytes. Perform fragmentation and include the pertinent values of the IP header of the original packet and of each fragment.

The maximum possible data length per fragment = $576 - 20 = 556$ bytes. However, 556 is not a multiple of 8. Thus we need to set the maximum data length to 552 bytes. We can break 1484 into $552 + 552 + 380$ (other combinations are also possible).

Table 8.1 shows the pertinent values for the IP header where x denotes a unique identification value. Other values, except the header checksum, are the same as in the original packet.

TABLE 8.1 Values of the IP header in a fragmented packet.

	Total length	ID	MF	Fragment offset
Original packet	1504	x	0	0
Fragment 1	572	x	1	0
Fragment 2	572	x	1	69
Fragment 3	400	x	0	138

8.2.9 ICMP: Error and Control Messages

It was noted earlier that if a router could not forward a packet for some reason (for example, the TTL value reaches 0, or the packet length is greater than the network MTU while the DF bit is set), the router would send an error message back to the source to report the problem. The **Internet Control Message Protocol (ICMP)** is the protocol that handles error and other control messages. Although ICMP messages are encapsulated by IP packets (with protocol number 1), ICMP is considered to be in

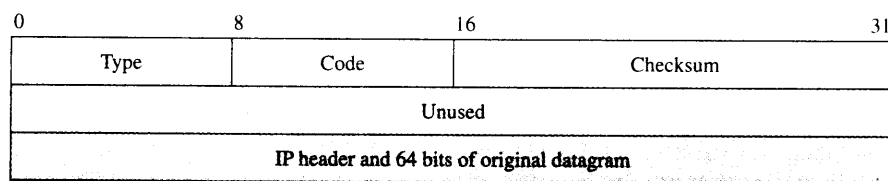


FIGURE 8.12 ICMP basic error message format.

the same layer as IP. Each ICMP message format begins with a type field to identify the message. Some ICMP message types are echo reply, destination unreachable, source quench, redirect, echo request, time exceeded, parameter problem, timestamp request, and timestamp reply. We describe several important types in this section (detailed information can be found in [RFC 792]).

Basic error messages generally follow the format shown in Figure 8.12. The description of each field follows.

Type: This field identifies the type of the message. Several types are described below and more detailed information can be found in [RFC 792].

Code: For a given type, the code field describes the purpose of the message.

Checksum: This field is used to detect errors in the ICMP message.

IP header plus original datagram: This field can be used for diagnostic purposes by matching the information (e.g., port numbers) in the ICMP message with the original data in the IP packet.

For example, a *type 3* message (that is, with a type field value of 3) indicates a problem reaching the destination. The particular problem is identified by one of the following possible values for the code field:

- | | |
|------------------------|-----------------------------------|
| 0 Network Unreachable | 3 Port Unreachable |
| 1 Host Unreachable | 4 Fragmentation needed and DF set |
| 2 Protocol Unreachable | 5 Source route failed |

A *type 11* message indicates a time-exceeded problem. A code field of 0 indicates that the TTL value has been exceeded. A code field of 1 indicates that the fragment reassembly time has been exceeded. The ICMP time-exceeded message is exploited in the Traceroute program. When a packet reaches a router with the value of the TTL equal to 0 or 1 before the packet reaches the destination, the corresponding router will send an ICMP message with type "time exceeded" back to the originating host. The time-exceeded message also contains the IP address of the router that issues the message. Thus, by sending messages to the destination with the TTL incremented by one per message, a source host will be able to trace the sequence of routers to the destination.

Echo request and echo reply messages follow the format shown in Figure 8.13. When a destination receives an echo request message from a source, the destination simply replies with a corresponding echo reply message back to the source. The echo request and echo reply messages are used in the PING program and are often used to determine whether a remote host is alive. PING is also often used to estimate the round-trip time between two hosts.

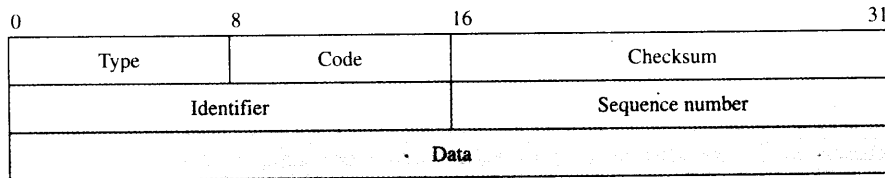


FIGURE 8.13 Echo request and echo reply message format.

Type 8 is used for echo request while type 0 for echo reply. The code field is set to zero for both types. The sequence number field is used to match the echo reply message with the corresponding echo request message. The identifier field can be used to differentiate different sessions using the echo services. The data from the echo request message is simply copied in the echo reply message and can be used for diagnostic purposes. The data field is of variable length.

Figure 8.14 and Figure 8.15 show the use of ICMP packets to PING the host tesla.comm.utoronto.ca. The top pane of Figure 8.14 shows that frame 4 contains the first ICMP echo request packet. The middle pane shows the details of ICMP packet fields. The type 8 indicates that this is an ICMP echo request. The identifier (0x0200) and the sequence number (5b:00) are included to help identify the echo reply. The middle pane in Figure 8.15 shows frame 5 contains the first ICMP echo reply packet indicated by the type 0 and by identifier and sequence number fields that match the first request. Receipt of the echo reply verifies that a valid path between the two hosts exists.

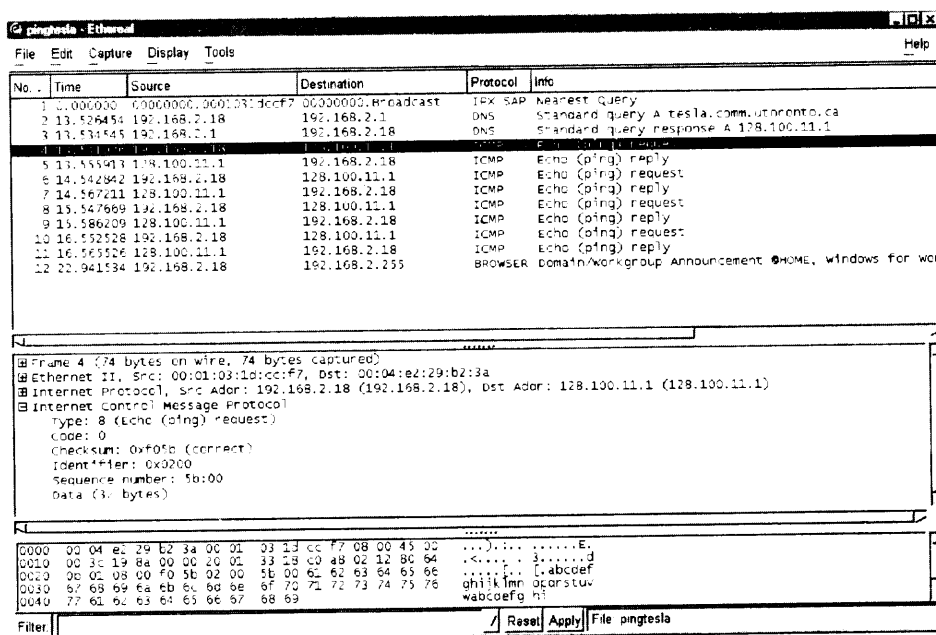


FIGURE 8.14 Example of ICMP echo request in PING.

The screenshot shows a network traffic capture window with a list of packets and a detailed view of a selected packet.

No.	Time	Source	Destination	Protocol	Info
1	0.000000	00000000.0001031d cc f7	00000000.Broadcast	IPX SAP	Nearest Query
2	13.526454	192.168.2.18	192.168.2.1	DNS	Standard query A tesla.comm.utoronto.ca
3	13.534545	192.168.2.1	192.168.2.18	DNS	Standard query response A 128.100.11.1
4	13.541026	192.168.2.18	128.100.11.1	ICMP	Echo (ping) request
5	14.542842	128.100.11.1	192.168.2.18	ICMP	Echo (ping) reply
6	14.542842	192.168.2.18	128.100.11.1	ICMP	Echo (ping) request
7	14.567211	128.100.11.1	192.168.2.18	ICMP	Echo (ping) reply
8	15.547669	192.168.2.18	128.100.11.1	ICMP	Echo (ping) request
9	15.586209	128.100.11.1	192.168.2.18	ICMP	Echo (ping) reply
10	16.552528	192.168.2.18	128.100.11.1	ICMP	Echo (ping) request
11	16.565526	128.100.11.1	192.168.2.18	ICMP	Echo (ping) reply

Frame 5 (74 bytes on wire, 74 bytes captured)
 Ethernet II, Src: 00:04:e2:29:b2:3a, Dst: 00:01:03:1d:cc:f7
 Internet Protocol, Src Addr: 128.100.11.1 (128.100.11.1), Dst Addr: 192.168.2.18 (192.168.2.18)
 Internet Control Message Protocol
 Type: 0 (Echo (ping) reply)
 Code: 0
 Checksum: 0xF85b (correct)
 Identifier: 0x0200
 Sequence number: 5b:00
 Data (32 bytes)

```

0000 00 01 03 1d cc f7 00 04 e2 29 b2 3a 08 00 45 00 .....E.
0010 00 3c 99 88 00 00 f0 01 e3 18 80 64 0b 01 c0 a8 .....d...
0020 02 12 00 00 f8 5b 02 00 5b 00 61 62 63 64 65 66 .....[.abcdef
0030 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 ghijklmnoprstuv
0040 77 61 62 63 64 65 66 67 68 69 .....wabcdefg hi
  
```

Filter: Reset Apply File: pingtesta

FIGURE 8.15 Example of ICMP echo reply in PING.

8.3 IPv6

IP version 4 has played a central role in the internetworking environment for many years. It has proved flexible enough to work on many different networking technologies. However, it has become a victim of its own success—explosive growth! In the early days of the Internet, people using it were typically researchers and scientists working in academia, high-tech companies, and research laboratories, mainly for the purpose of exchanging scientific results through e-mails. In the 1990s the World Wide Web and personal computers shifted the user of the Internet to the general public. This change has created heavy demands for new IP addresses, and the 32 bits of the current IP addresses will be exhausted sooner or later.

In the early 1990s the Internet Engineering Task Force (IETF) began to work on the successor of IP version 4 that would solve the address exhaustion problem and other scalability problems. After several proposals were investigated, a new IP version was recommended in late 1994. The new version is called **IPv6** for IP version 6 (also called *IP next generation* or *IPng*) [RFC 2460]. IPv6 was designed to interoperate with IPv4 since it would likely take many years to complete the transition from version 4 to version 6. Thus IPv6 should retain the most basic service provided by IPv4—a connectionless delivery service. On the other hand, IPv6 should also change the IPv4 functions that do not work well and support new emerging applications such as real-time video conferencing, etc. Some of the changes from IPv4 to IPv6 include:

Longer address fields: The length of address field is extended from 32 bits to 128 bits. The address structure also provides more levels of hierarchy. Theoretically, the address space can support up to 3.4×10^{38} hosts.

Simplified header format: The header format of IPv6 is simpler than that of IPv4.

Some of the header fields in IPv4 such as checksum, IHL, identification, flags, and fragment offset do not appear in the IPv6 header.

Flexible support for options: The options in IPv6 appear in optional *extension headers* that are encoded in a more efficient and flexible fashion than they are in IPv4.

Flow label capability: IPv6 adds a “flow label” to identify a certain packet “flow” that requires a certain QoS.

Security: IPv6 supports built-in authentication and confidentiality.

Large packets: IPv6 supports payloads that are longer than 64 K bytes, called *jumbo* payloads.

Fragmentation at source only: Routers do not perform packet fragmentation. If a packet needs to be fragmented, the source should check the minimum MTU along the path and perform the necessary fragmentation.

No checksum field: The checksum field has been removed to reduce packet processing time in a router. Packets carried by the physical network such as Ethernet, token ring, or X.25 are typically already checked. Furthermore, higher-layer protocols such as TCP and UDP also perform their own verification. Thus the removal of the checksum field is unlikely to introduce a serious problem in most situations.

8.3.1 Header Format

The IPv6 header consists of a required basic header and optional extension headers. The format of the basic header is shown in Figure 8.16. The packet should be transmitted in network byte order.

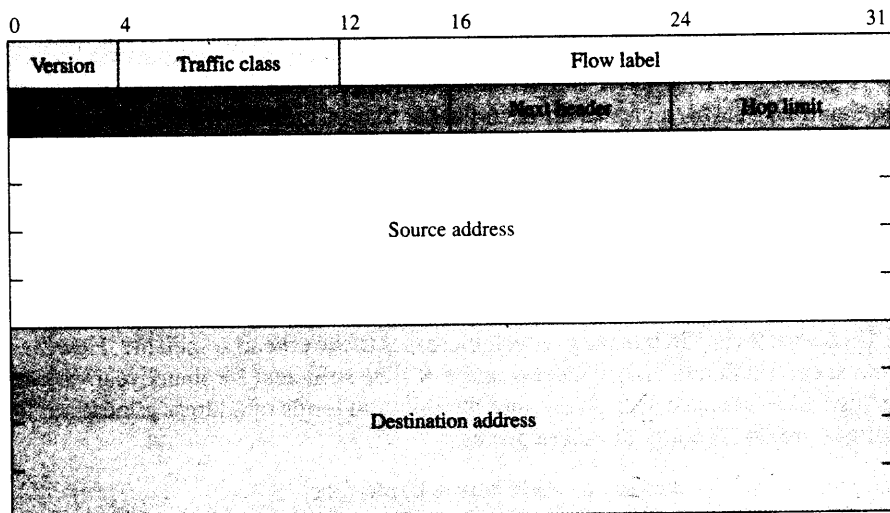


FIGURE 8.16 IPv6 basic header.

The description of each field in the basic header follows.

Version: The version field specifies the version number of the protocol and should be set to 6 for IPv6. The location and length of the version field stays unchanged so that the protocol software can recognize the version of the packet quickly.

Traffic class: The traffic class field specifies the traffic class or priority of the packet. The traffic class field is intended to support differentiated service.

Flow label: The flow label field can be used to identify the QoS requested by the packet. In the IPv6 standard, a flow is defined as “a sequence of packets sent from a particular source to a particular (unicast or multicast) destination for which the source desires special handling by the intervening routers.” An example of an application that may use a flow label is a packet video system that requires its packets to be delivered to the destination within a certain time constraint. Routers that see these packets will have to process them according to their QoS. Hosts that do not support flows are required to set this field to 0.

Payload length: The payload length indicates the length of the data (excluding header). With 16 bits allocated to this field, the payload length is limited to 65,535 bytes. As we explain below, it is possible to send larger payloads by using the option in the extension header.

Next header: The next header field identifies the type of the extension header that follows the basic header. The extension header is similar to the options field in IPv4 but is more flexible and efficient. Extension headers are further discussed below.

Hop limit: The hop limit field replaces the TTL field in IPv4. The name now says what it means: The value specifies the number of hops the packet can travel before being dropped by a router.

Source address and destination address: The source address and the destination address identify the source host and the destination host, respectively. The address format is discussed below.

Figure 8.17 gives an example of IPv6 packet exchanges. A comparison with Figure 8.5 shows that IPv6 has a simpler header structure. The middle pane zooms in on frame number 7, which carries an IPv6 packet that in turn carries a DNS query. The version field indicates the IPv6 is involved, that the payload length is 39 bytes, and that the next header is a UDP header.

8.3.2 Network Addressing

The IPv6 address is 128 bits long, which increases the overhead somewhat. However, it is almost certain that the huge address space will be sufficient for many years to come. The huge address space also gives more flexibility in terms of address allocation. IPv6 addresses are divided into three categories:

1. *Unicast* addresses identify a single network interface.
2. *Multicast* addresses identify a group of network interfaces, typically at different locations. A packet will be sent to all network interfaces in the group.

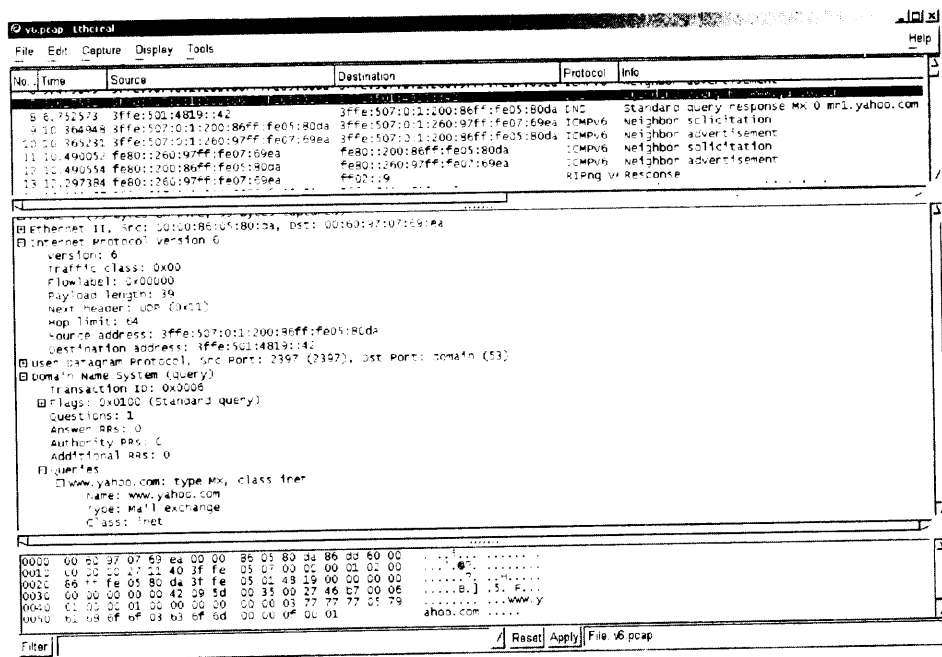


FIGURE 8.17 Example of an IPv6 packet header.

3. *Anycast* addresses also identify a group of network interfaces. However, a packet will be sent to only one network interface in the group, usually the nearest one.

Note that a broadcast address can be supported with a multicast address by making the group consist of all interfaces in the network.

Recall that the IPv4 address typically uses the dotted-decimal notation when communicated by people. It should become obvious that the dotted-decimal notation can be rather long when applied to IPv6 long addresses. A more compact notation that is specified in the standard is to use a hexadecimal digit for every 4 bits and to separate every 16 bits with a colon. An example of an IPv6 address is

4BF5:AA12:0216:FEBC:BA5F:039A:BE9A:2176

Often IPv6 addresses can be shortened to a more compact form. The first shorthand notation can be exploited when the 16-bit field has some leading zeros. In this case the leading zeros can be removed, but there must be at least one numeral in the field. As an example

4BF5:0000:0000:0000:BA5F:039A:000A:2176

can be shortened to

4BF5:0:0:0:BA5F:39A:A:2176

Further shortening is possible where consecutive zero-valued fields appear. These fields can be shortened with the double-colon notation (::). Of course, the double-colon notation can appear only once in an address, since the number of zero-valued fields is not encoded and needs to be deduced from the specified total number of fields. Continuing with the preceding example, the address can be written even more compactly as

4BF5::BA5F:39A:A:2176

To recover the original address from one containing a double colon, you take the nonzero values that appear to the left of the double colons and align them to the left. You then take the number that appears to the right of the double colons and align them to the right. The field in between is set to 0s.

The dotted-decimal notation of IPv4 can be mixed with the new hexadecimal notation. This approach is useful for the transition period when IPv4 and IPv6 coexist. An example of a mixed notation is

::FFFF:128.155.12.198

Address allocations are organized by types, which are in turn classified according to prefixes (leading bits of the address). At the time of this writing, the initial allocation for prefixes is given in Table 8.2.

Less than 30 percent of the address space has been assigned. The remaining portion of the address space is for future use. Most types are assigned for unicast addresses, except the one with a leading byte of 1s, which is assigned for multicast. Anycast addresses are not differentiated from unicast and share the same address space.

TABLE 8.2 Address types based on prefixes.

Binary prefix	Types	Percentage of address space
0000 0000	Reserved	0.39
0000 0001	Unassigned	0.39
0000 001	ISO network addresses	0.78
0000 010	IPX network addresses	0.78
0000 011	Unassigned	0.78
0000 1	Unassigned	3.12
0001	Unassigned	6.25
001	Unassigned	12.5
010	Provider-based unicast addresses	12.5
011	Unassigned	12.5
100	Geographic-based unicast addresses	12.5
101	Unassigned	12.5
110	Unassigned	12.5
1110	Unassigned	6.25
1111 0	Unassigned	3.12
1111 10	Unassigned	1.56
1111 110	Unassigned	0.78
1111 1110 0	Unassigned	0.2
1111 1110 10	Link local use addresses	0.098
1111 1110 11	Site local use addresses	0.098
1111 1111	Multicast addresses	0.39

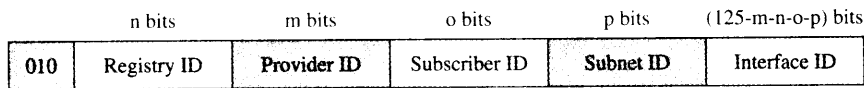


FIGURE 8.18 Provider-based address format.

IPv6 assigns a few addresses for special purposes. The address 0::0 is called the *unspecified address* and is never used as a destination address. However, it may be used as a source address when the source station wants to learn its own address. The address ::1 is used for a loopback whose purpose is the same as the loopback address in IPv4. Another set of special addresses is needed during the transition period where an IPv6 packet needs to be “tunneled” across an IPv4 network. These addresses, called *IPv4-compatible* addresses, are used by IPv6 routers and hosts that are directly connected to an IPv4 network. The address format consists of 96 bits of 0s followed by 32 bits of IPv4 address. Thus an IPv4 address of 135.150.10.247 can be converted to an IPv4-compatible IPv6 address of ::135.150.10.247. A similar set of special addresses is used to indicate IPv4 hosts and routers that do not support IPv6. These addresses are called *IP-mapped* addresses. The format of these addresses consists of 80 bits of 0s, followed by 16 bits of 1s, and then by 32 bits of IPv4 address.

Provider-based unicast addresses are identified by the prefix 010. It appears that these addresses will be mainly used by the Internet service providers to assign addresses to their subscribers. The format of these addresses is shown in Figure 8.18.

Notice the hierarchical structure of this address format. The first level is identified by the registry ID, which is managed by ARIN (North America), RIPE (Europe), or APNIC (Asia Pacific). The next level identifies the Internet service provider that is responsible for assigning the subscriber IDs. Finally, each subscriber assigns the addresses according to the subnet IDs and interface IDs.

The local addresses are used for a collection of hosts that do not want to connect to the global Internet because of security and privacy concerns. There are two types of local addresses: link-local addresses and site-local addresses. The link-local addresses are used for a single link, while the site-local addresses are used for a single site. The local addresses are designed so that when an organization decides to connect the hosts to the global Internet, the move will be as painless as possible.

8.3.3 Extension Headers

To support extra functionalities that are not provided by the basic header, IPv6 allows an arbitrary number of extension headers to be placed between the basic header and the payload. Extension headers act like options in IPv4 except the former are encoded more efficiently and flexibly, as we show soon. The extension headers are daisy chained by the next header field, which appears in the basic header as well as in each extension header. Figure 8.19 illustrates the use of the next header field. A consequence of the daisy-chain formation is that the extension headers must be processed in the order in which they appear in the packet.

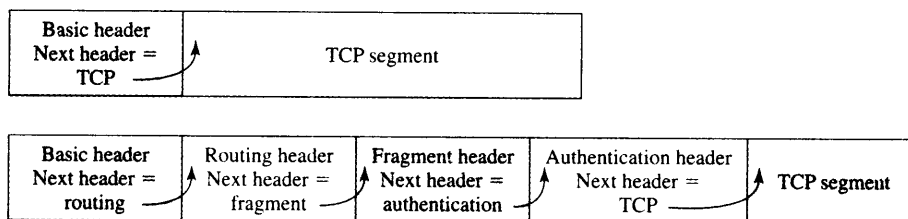


FIGURE 8.19 Daisy-chain extension headers.

Six extension headers have been defined. They are listed in Table 8.3. These extension headers should appear in a packet as they are listed in the table from top to bottom.⁵

LARGE PACKET

IPv6 allows a payload size of more than 64K by using an extension header. The use of payload size greater than 64K has been promoted mainly by people who work on supercomputers. Figure 8.20 shows the format of the extension header for a packet with a jumbo payload (length that is greater than 65,535 bytes). The next header field identifies the type of header immediately following this header. The value 194 defines a jumbo payload option. The payload length in the basic header must be set to 0. The option length field (opt len) specifies the size of the jumbo payload length field in bytes. Finally, the jumbo payload length field specifies the payload size. With 32 bits the payload size can be as long as 4,294,967,295 bytes.

FRAGMENTATION

As noted previously, IPv6 allows only the source host to perform fragmentation. Intermediate routers no longer need to perform fragmentation. If the packet length is greater than the MTU of the network this packet is to be forwarded to, an intermediate router discards the packet and sends an ICMP error message back to the source. A source can find the minimum MTU along the path from the source to the destination by performing a “path MTU discovery” procedure. An advantage of doing fragmentation at the source only is that routers can process packets faster, which is important in a high-speed environment. A disadvantage is that the path between a source and a

TABLE 8.3 Extension headers.

Header code	Header type
0	Hop-by-hop options header
43	Routing header
44	Fragment header
51	Authentication header
52	Encapsulating security payload header
60	Destination options header

⁵The authentication header and the encapsulating security payload header are discussed in Chapter 11.

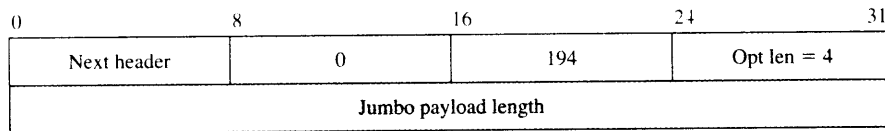


FIGURE 8.20 Extension header for jumbo packet.

destination must remain reasonably static so that the path MTU discovery does not give outdated information. If a source wants to fragment a packet, the source will include a fragment extension header (shown in Figure 8.21) for each fragment of the packet.

The fragment offset, M (more fragment), and identification fields have the same purposes as they have in IPv4 except the identification is now extended to 32 bits. Bits 8 to 15 and bits 29 and 30 are reserved for future use.

SOURCE ROUTING

Like IPv4, IPv6 allows the source host to specify the sequence of routers to be visited by a packet to reach the destination. This option is defined by a routing extension header, which is shown in Figure 8.22. The header length specifies the length of the routing extension header in units of 64 bits, not including the first 64 bits. Currently, only type 0 is specified. The segment left field identifies the number of route segments remaining before the destination is reached. Maximum legal value is 23. Initially, this value will be set to the total number of route segments from the source to the destination. Each router decrements this value by 1 until the packet reaches the destination. Each bit in the strict/loose bit mask indicates whether the next destination address must be followed strictly (if the bit is set to 1) or loosely (if the bit is set to 0).

8.3.4 Migration Issues from IPv4 to IPv6

Recall from Chapter 1 that the capability of a particular technology eventually saturates due to some fundamental limit. A new technology typically replaces an old one by providing new capabilities at the next level. We see that the addressing capabilities of IPv4 are reaching the saturation limit. IPv6 was developed to improve the addressing capabilities so that new devices requiring global addresses can be supported in the future. However, because IPv4 networks and hosts are widely deployed, migration issues need to be solved to ensure that the transition from IPv4 to IPv6 is as smooth as possible.

Current solutions are mainly based on the dual-IP-layer (or dual stack) approach whereby both IPv4 and IPv6 functions are present. For example, routers independently

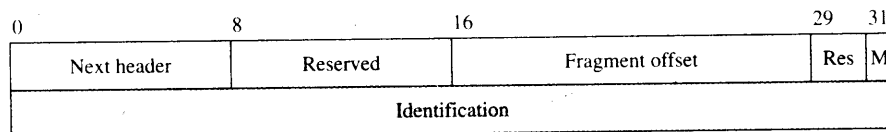


FIGURE 8.21 Fragment extension header.

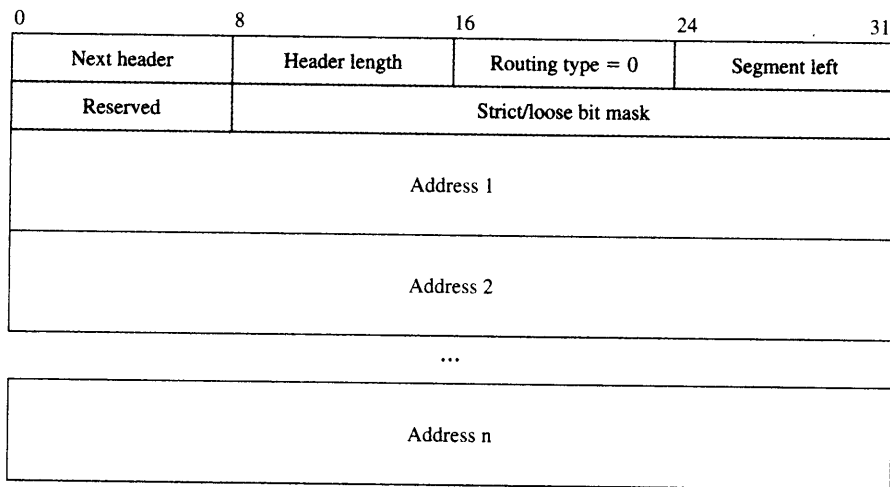


FIGURE 8.22 Routing extension header.

run both IPv4 and IPv6 routing protocols and can forward both types of packets. Recall that the type of a packet can be identified from the version field so that each incoming packet can be sent to the appropriate processing module.

When islands of IPv6 networks are separated by IPv4 networks, one approach is to build a tunnel across an IPv4 network connecting two IPv6 networks, as shown in Figure 8.23a. A **tunnel** is a path created between two nodes so that the tunnel appears as a single link to the user, as shown in Figure 8.23b. The tunneling approach essentially hides the route taken by the tunnel from the user. In our particular example, an IPv4 tunnel allows IPv6 packets to be forwarded across an IPv4 network without the IPv6 user having to worry about how packets are actually forwarded in the IPv4 network. A

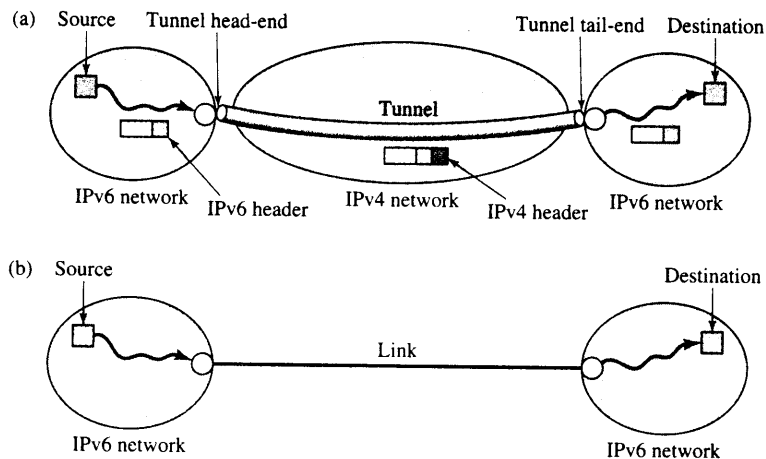


FIGURE 8.23 Tunneling: (a) IPv6 over IPv4 tunnel; (b) IPv6 virtual topology.

tunnel is typically realized by encapsulating each user packet in another packet that can be forwarded along the tunnel. In our example, IPv6 packets are first forwarded from the source to the tunnel head-end in the IPv6 network. At the tunnel head-end packets are encapsulated into IPv4 packets. Then IPv4 packets are forwarded in the IPv4 network to the tunnel tail-end where the reverse process (i.e., decapsulation) is performed. Finally, IPv6 packets are forwarded from the tunnel tail-end to the destination. Note that the tunnel endpoint can be located at a source host, an intermediate router, or a destination host. Note also that the concept of tunneling can be made recursive so that a tunnel may provide forwarding service to another tunnel.

8.4 USER DATAGRAM PROTOCOL

Two transport layer protocols, TCP and UDP, build on the best-effort service provided by IP to support a wide range of applications. In this section we discuss the details of UDP.

The **User Datagram Protocol (UDP)** is an unreliable, connectionless transport layer protocol. It is a very simple protocol that provides only two additional services beyond IP: demultiplexing and error checking on data. Recall that IP knows how to deliver packets to a host, but does not know how to deliver them to the specific application in the host. UDP adds a mechanism that distinguishes among multiple applications in the host. Recall also that IP checks only the integrity of its header. UDP can optionally check the integrity of the entire UDP datagram. Applications that do not require zero packet loss such as in packet voice systems are well suited to UDP. In practice, applications that use UDP include Trivial File Transfer Protocol, DNS, SNMP, and Real-Time Protocol (RTP).

The format of the UDP datagram is shown in Figure 8.24. The destination port allows the UDP module to demultiplex datagrams to the correct application in a given host. The source port identifies the particular application in the source host to receive replies. The UDP length field indicates the number of bytes in the UDP datagram (including header and data).

The UDP checksum field detects errors in the datagram, and its use is optional. If a source host does not want to compute the checksum, the checksum field should contain all 0s so that the destination host knows that the checksum has not been computed. What if the source host does compute the checksum and finds that the result is 0? The

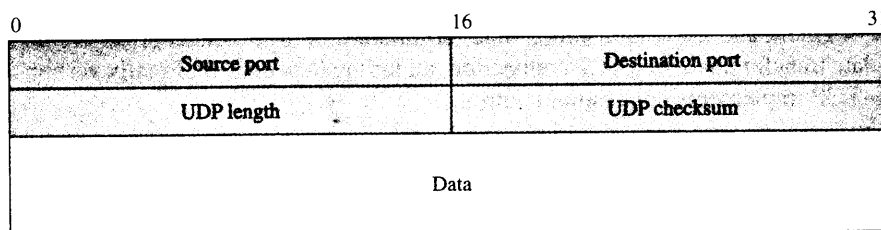


FIGURE 8.24 UDP datagram.

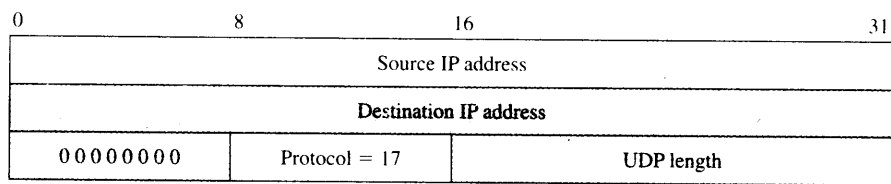


FIGURE 8.25 UDP pseudoheader.

algorithm that a host uses to compute the checksum will give a checksum field of all 1s. This is another representation of zero in 1s complement. The checksum computation procedure is similar to that in computing IP checksum except for two new twists. First, if the length of the datagram is not a multiple of 16 bits, the datagram will be padded out with 0s to make it a multiple of 16 bits. In doing so, the actual UDP datagram is not modified. The pad is used only in the checksum computation and is not transmitted. Second, UDP adds a **pseudoheader** (shown in Figure 8.25) to the beginning of the datagram when performing the checksum computation. The pseudoheader is also created by the source and destination hosts only during the checksum computation and is not transmitted. The pseudoheader is to ensure that the datagram has indeed reached the correct destination host and port. Finally, if a datagram is found to be corrupted, it is simply discarded and the source UDP entity is not notified.

8.5 TRANSMISSION CONTROL PROTOCOL

TCP [RFC 793] and IP are the workhorses of the Internet. The **Transmission Control Protocol (TCP)** provides a logical full-duplex (two-way) connection between two application layer processes across the Internet. TCP provides these application processes with a connection-oriented, reliable, in-sequence, byte-stream service. TCP also provides flow control that allows a TCP receiver to control the rate at which the sender transmits information so that the receiver buffers do not overflow. In addition, TCP also provides congestion control that induces senders to reduce the rate at which they send packets when there is congestion in the routers. TCP can support multiple application processes in the same end system.

In Chapter 2 we discussed the role of TCP in supporting application layer protocols. In Section 5.3 we showed how TCP provides reliable byte stream service using a form of Selective Repeat ARQ. In this section we first summarize the basic operation of TCP and the reliable byte stream service it provides. We then discuss the format of the TCP segment and the establishment and termination of TCP connections. We discuss the data transfer phase of a TCP connection, including flow control. Finally we consider how TCP implements congestion control.

8.5.1 TCP Operation and Reliable Stream Service

A TCP connection goes through the three phases of a connection-oriented service. The *TCP connection establishment phase* sets up a connection between the two application

processes by creating and initializing variables that are used in the protocol. These variables are stored in a connection record that is called the **transmission control block (TCB)**. Once the connection is established, TCP enters the *data transfer phase* where it delivers data over each direction in the connection correctly and in sequence. TCP was designed to operate over IP and does not assume that the underlying network service is reliable. To implement reliability, TCP uses a form of Selective Repeat ARQ. When the applications are done exchanging data, TCP enters the *connection termination phase* where each direction of the connection is terminated independently, allowing data to continue flowing in one direction after the other direction has been closed.

As indicated in Chapter 2, a TCP connection is uniquely identified by four parameters: the sender IP address and port number, and the destination IP address and port number. An end system can therefore support multiple simultaneous TCP connections. Typically the server is assigned a well-known port number and the client is assigned an ephemeral port number that is selected from a pool of available numbers when the connection is set up.

As shown in Figure 8.26, the application layer writes the data it needs to transmit into a buffer. TCP does not preserve message boundaries and treats the data it gets from the application layer as a byte stream. Thus when a source writes a 1000-byte message in a single chunk (one write), the destination may receive the message in two chunks of 500 bytes each (two reads), in three chunks of 400 bytes, 300 bytes and 300 bytes, or in any other combination. In other words, TCP may split or combine the application information and pack it into segments in the way it finds most appropriate for transfer over the underlying network.

The TCP transmitter arranges a consecutive string of bytes into a segment. The segment contains a header with address information that enables the network to direct the segment to its destination application process. The segment contains a *sequence number* that corresponds to the number of the first byte in the string that is being transmitted. The segment also contains an Internet checksum. The TCP receiver performs an error check on each segment it receives. If the segment is error-free and is not a duplicate segment, the receiver inserts the bytes into the appropriate locations in the receive buffer if the bytes fall within the receive window. The receiver will accept out-of-order but error-free segments, so the receive buffer can have gaps where bytes have not been received. The receiver keeps track of the oldest byte it has not yet received. It sends the sequence number of this byte in the acknowledgments it sends back to the transmitter. The receive window slides forward whenever the desired next oldest byte is received. The TCP transmitter uses a time-out mechanism to trigger retransmissions

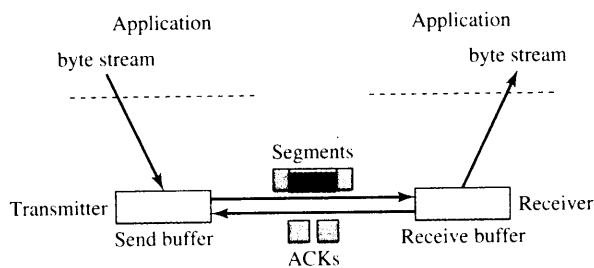


FIGURE 8.26 TCP preview.

when a segment is not acknowledged within a certain time. The TCP transmitter send window slides forward whenever the acknowledgments for pending bytes are received.

TCP was designed to deliver a connection-oriented service in the Internet environment, where different IP packets may traverse a different path from the same source to the same destination and may therefore arrive out of order. Therefore, it is possible for old segments from previous connections to arrive at a receiver, thus potentially complicating the task of eliminating duplicate segments. TCP deals with this problem by using long sequence numbers (32 bits) and by establishing randomly selected initial sequence numbers during connection setup. At any given time the receiver is accepting sequence numbers (bytes) from a much smaller window, so the likelihood of accepting a very old message is very low. In addition, TCP enforces a time-out period at the end of each connection to allow the network to clear old segments from the network.

TCP separates the flow control function from the acknowledgment function. The flow control function is implemented through an *advertised window* field in the segment header. Segments that travel in the reverse direction contain the advertised window size that informs the transmitter of the number of bytes that can be currently accommodated in the receiver buffers.

MAXIMUM SEGMENT LIFETIME, REINCARNATION, AND THE Y2K BUG

The Y2K bug is the result of an ambiguity that arises from the limited precision used in early computer programs to specify the calendar year. The use of only two decimal digits to represent a year results in an ambiguity between the year 1900 and the year 2000. Consequently, there were many concerns about unanticipated actions that might have been taken by these programs when the year 2000 was reached. This same problem is faced by millions of TCP processes every second of every day.

A TCP connection is identified by the source and destination port numbers and by the IP address of the source and destination machines. During its lifetime, the TCP connection will send some number of segments using the 32-bit byte sequence numbering. Each segment is encapsulated in an IP packet and sent into the Internet. It is possible for an IP packet to get trapped in a loop inside the network, typically while the routing tables adapt to a link or router failure. Such a packet is called a *lost* or *wandering duplicate*. In the meantime TCP at the sending side times out and sends a retransmission of the segment that arrives promptly using the new route. If the wandering duplicate subsequently arrives at the same connection, then the segment will be recognized as a duplicate and be rejected. (This scenario assumes that segments are not being sent so fast that the sequence numbers have not already wrapped around.)

It is also possible for the TCP connection to end, even while one of its wandering duplicates is still in the network. Suppose that a new TCP connection is set up between the same two machines and with the same port numbers. The new TCP connection is called an *incarnation* of the previous connection. TCP needs to protect the new connection so that duplicates from previous connections are

prevented from being accepted and interpreted by the new connection. For example, the duplicate could be the command to terminate a connection. To deal with these and other problems, we show in the next section that every implementation of TCP assumes a certain value for the **maximum segment lifetime (MSL)**, which is the maximum time that an IP packet can live in the network.

8.5.2 TCP Protocol

We now discuss the structure of TCP segments and the setting up of a TCP connection, the data transfer phase, and the closing of the connection. Detailed information about TCP can be found in RFC 793 and RFC 1122.

TCP SEGMENT

Figure 8.27 shows the format of the TCP segment. The header consists of a 20-byte fixed part plus a variable-size options field.

The description of each field in the TCP segment is given below. The term *sender* refers to the host that sends the segment, and *receiver* refers to the host that receives the segment.

Source port and destination port: The source and destination ports identify the sending and receiving applications, respectively. Recall from Section 2.3.2 that the pair of ports and IP addresses identify a process-to-process connection.

Sequence number: The 32-bit sequence number field identifies the position of the first data byte of this segment in the sender's byte stream during data transfer (when SYN bit is not set). The sequence number wraps back to 0 after $2^{32} - 1$. Note that TCP identifies the sequence number for each byte (rather than for each

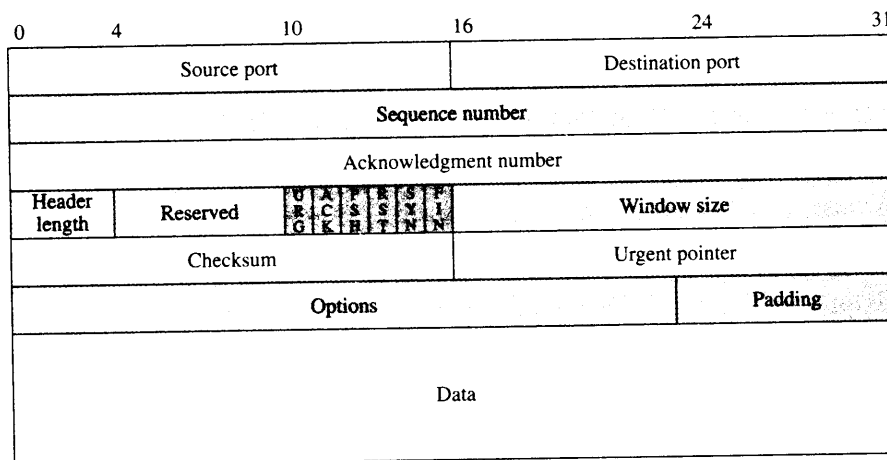


FIGURE 8.27 TCP segment.

segment). For example, if the value of the sequence number is 100 and the data area contains five bytes, then the next time this TCP module sends a segment, the sequence number will be 105. If the SYN bit is set to 1 (during connection establishment), the sequence number indicates the **initial sequence number (ISN)** to be used in the sender's byte stream. The sequence number of the first byte of data for this byte stream will be $ISN + 1$. It is important to note that a TCP connection is full duplex so that each end point independently maintains its own sequence number.

Acknowledgment number: This field identifies the sequence number of the next data byte that the sender expects to receive if the ACK bit is set. This field also indicates that the sender has successfully received all data up to but not including this value. If the ACK bit is not set (during connection establishment), this field is meaningless. Once a connection is established, the ACK bit must be set.

Header length: This field specifies the length of the TCP header in 32-bit words. This information allows the receiver to know the beginning of the data area because the options field is variable length.

Reserved: As the name implies, this field is reserved for future use and must be set to 0.

URG: If this bit is set, the urgent pointer is valid (discussed shortly).

ACK: If this bit is set, the acknowledgment number is valid.

PSH: When this bit is set, it tells the receiving TCP module to pass the data to the application immediately. Otherwise, the receiving TCP module may choose to buffer the segment until enough data accumulates in its buffer.

RST: When this bit is set, it tells the receiving TCP module to abort the connection because of some abnormal condition.

SYN: This bit requests a connection (discussed later).

FIN: When this bit is set, it tells the receiver that the sender does not have any more data to send. The sender can still receive data from the other direction until it receives a segment with the FIN bit set.

Window size: The window size field specifies the number of bytes the sender is willing to accept. This field can be used to control the flow of data and congestion.

Checksum: This field detects errors on the TCP segment. The procedure is discussed below.

Urgent pointer: When the URG bit is set, the value in the urgent pointer field added to that in the sequence number field points to the last byte of the "urgent data" (data that needs immediate delivery). However, the first byte of the urgent data is never explicitly defined. Because the receiver's TCP module passes data to the application in sequence, any data in the receiver's buffer up to the last byte of the urgent data may be considered urgent.

Options: The options field may be used to provide other functions that are not covered by the header. If the length of the options field is not a multiple of 32 bits, extra padding bits will be added. The most important option is used by the sender to indicate the **maximum segment size (MSS)** it can accept. This option is specified during connection setup. Two other options that are negotiated during connection setup are intended to deal with situations that involve large delay-bandwidth products. The **window scale** option allows the use of a larger

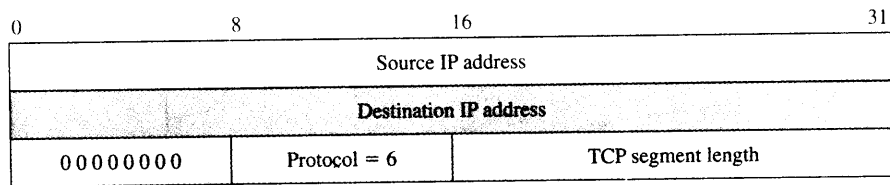


FIGURE 8.28 TCP pseudoheader.

advertised window size. The window can be scaled upward by a factor of up to 2^{14} . Normally the maximum window size is $2^{16} - 1 = 65,535$. With scaling the maximum advertised window size is $65,535 \times 2^{14} = 1,073,725,440$ bytes. The **timestamp** option is intended for high-speed connections where the sequence numbers may wrap around during the lifetime of the connection. The timestamp option allows the sender to include a timestamp in every segment. This timestamp can also be used in the RTT calculation.

TCP CHECKSUM

The purpose of the TCP checksum field is to detect errors. The checksum computation procedure is similar to that used to compute an IP checksum except for two features. First, if the length of the segment is not a multiple of 16 bits, the segment will be padded with zeros to make it a multiple of 16 bits. In doing so, the TCP length field is not modified. Second, a **pseudoheader** (shown in Figure 8.28) is added to the beginning of the segment when performing the checksum computation. The pseudoheader is created by the source and destination hosts during the checksum computation and is not transmitted. This mechanism ensures the receiver that the segment has indeed reached the correct destination host and port and that the protocol type is TCP (which is assigned the value 6). At the receiver the IP address information in the IP packet that contained the segment is used in the checksum calculation.

CONNECTION ESTABLISHMENT

Before any host can send data, a connection must be established. TCP establishes the connection using a **three-way handshake** procedure shown in Figure 8.29. The handshakes are described in the following steps:

1. Host A sends a connection request to host B by setting the SYN bit. Host A also registers its initial sequence number to use (Seq.no = x).
2. Host B acknowledges the request by setting the ACK bit and indicating the next data byte to receive (Ack.no = $x + 1$). The “plus one” is needed because the SYN bit consumes one sequence number. At the same time, host B also sends a request by setting the SYN bit and registering its initial sequence number to use (Seq.no = y).
3. Host A acknowledges the request from B by setting the ACK bit and confirming the next data byte to receive (Ack.no = $y + 1$). Note that the sequence number is set to $x + 1$. On receipt at B the connection is established.

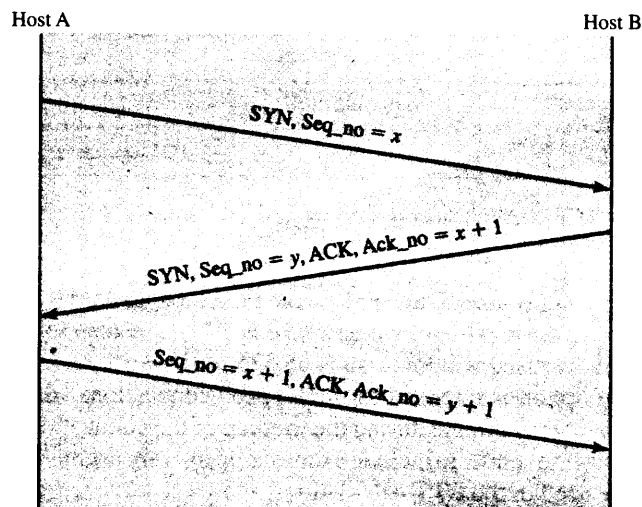


FIGURE 8.29 Three-way handshake.

If during a connection establishment phase, one of the hosts decides to refuse a connection request, it will send a reset segment by setting the RST bit. Each SYN message can specify options such as maximum segment size, window scaling, and timestamps.

Because TCP segments can be delayed, lost, and duplicated, the initial sequence number should be different each time a host requests a connection.⁶ The three-way handshake procedure ensures that both endpoints agree on their initial sequence numbers. To see why the initial sequence number must be different, consider a case in which a host can always use the same initial sequence number, say, n , as shown in Figure 8.30. After a connection is established, a delayed segment from the previous connection arrives. Host B accepts this segment, since the sequence number turns out to be legal. If a segment from the current connection arrives later, it will be rejected by host B, thinking that the segment is a duplicate. Thus host B cannot distinguish a delayed segment from the new one. The result can be devastating if the delayed segment says, for example, "Transfer 1 million dollars from my account." You should verify that if the initial sequence number is always unique, the delayed segment is very unlikely to possess a legal sequence number and thus can be detected and discarded.

An example of the segments exchanged to carry out the TCP three-way handshake was already presented in Chapter 5. Figure 8.31 shows the details of the second TCP segment in the three-way handshake. This segment has the ACK set to acknowledge the receipt of the first SYN segment as well as the ACK sequence number 1839733356. The segment also has SYN set to request the connection in the opposite direction with initial sequence number 1877388864. Finally, the segment indicates a window size of 49152 bytes and it indicates that its maximum segment size is 1460 bytes as well.

⁶The specification recommends incrementing the initial sequence number by 1 every four microseconds.

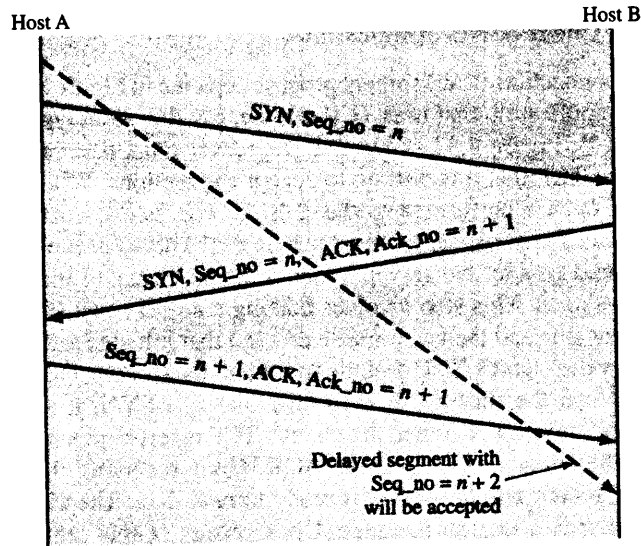


FIGURE 8.30 Justifying a three-way handshake: If a host always uses the same initial sequence number, old segments cannot be distinguished from the current ones.

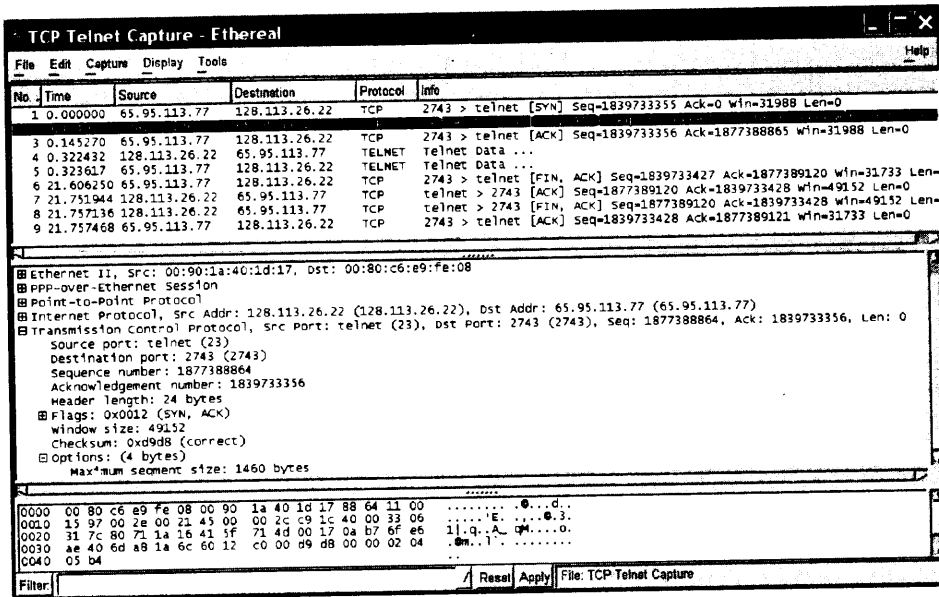


FIGURE 8.31 Example of TCP segment.

EXAMPLE A Client/Server Application

Let us revisit the connection establishment process depicted in Figure 8.29 in the context of a client/server application that uses TCP service. Let the client reside in host A and the server in host B. Figure 8.32 shows that the server must first carry out a *passive open* to indicate to TCP that it is willing to accept connections. When using Berkeley sockets, a passive open is performed by the calls `socket`, `bind`, `listen`, and `accept`. Recall from Chapter 2 that the server must use its well-known port number so that the client knows how to contact the server. When a client wishes to initiate a session, it performs an *active open*. This step involves making a `socket` call (t_1) that creates the socket on the client side and then a `connect` call (t_2) that initiates the TCP connection. This action causes the client's TCP module to initiate the three-way handshake shown in Figure 8.29. When the server's TCP receives the first SYN, it returns a segment with an ACK and its own SYN. When the client's TCP receives this segment, `connect` returns (t_3) and the client's TCP sends an ACK. Upon receiving this ACK, `accept` returns (t_4) in the server, and the server is ready to read data. The client then issues a `write` call (t_5) to send a request message. Upon receipt of this segment by the TCP module in the server, `read` returns (t_6), and the request message is passed to the server. Subsequently, the server sends a reply message.

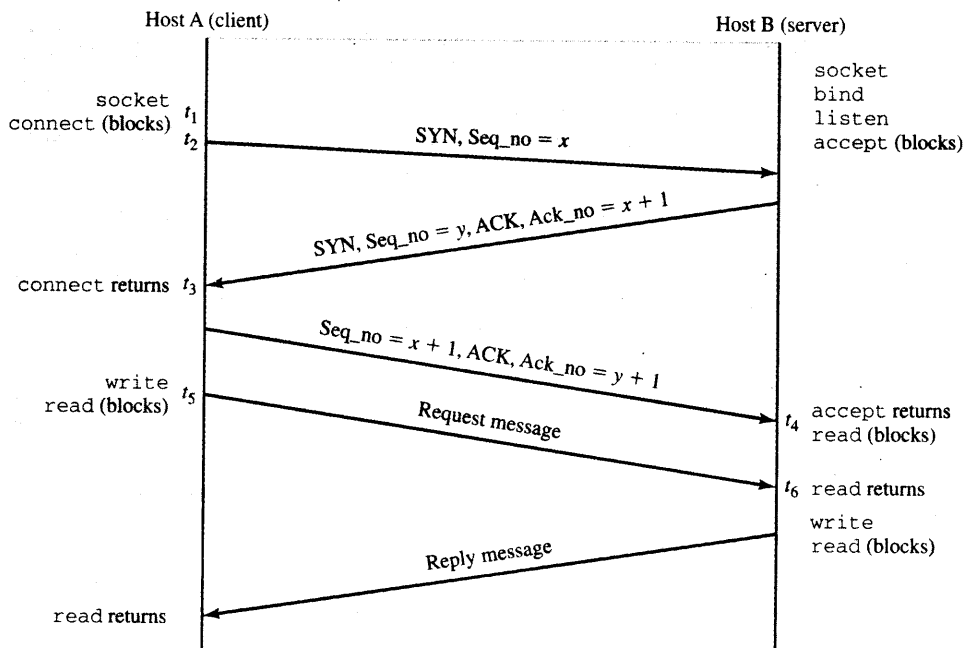


FIGURE 8.32 Client/server application process actions and TCP.

DATA TRANSFER

To provide a reliable delivery service to applications, TCP uses the Selective Repeat ARQ protocol with positive acknowledgment implemented by a sliding-window mechanism. The difference here is that the window slides on a byte basis instead of on a packet basis. TCP can also apply flow control over a connection by dynamically advertising the window size. Flow control is the process of regulating the traffic between two points and is used to prevent the sender from overwhelming the receiver with too much data.

Figure 8.33 illustrates an example of how a TCP entity can exert flow control. Suppose that at time t_0 , the TCP module in host B advertised a window of size 2048 and expected the next byte received to have a sequence number 2000. The advertised window allows host A to transmit up to 2048 bytes of unacknowledged data. At time t_1 , host A has only 1024 bytes to transmit, so it transmits all the data starting with the sequence number 2000. The TCP entity also advertises a window of size 1024 bytes to host B, and the next byte is expected to have a sequence number of 1. When the segment arrives, host B chooses to delay the acknowledgment in the hope that the acknowledgment can ride freely with the data. Meanwhile at time t_2 , A has another 1024 bytes of data and transmits it. After the transmission, A's sending window closes completely. It is not allowed to transmit any more data until an acknowledgment comes back.

At time t_3 , host B has 128 bytes of data to transmit; it also wants to acknowledge the first two segments of data from host A. Host B can simply **piggyback** the acknowledgment (by specifying the acknowledgment number to be 4048) to the data segment. Also

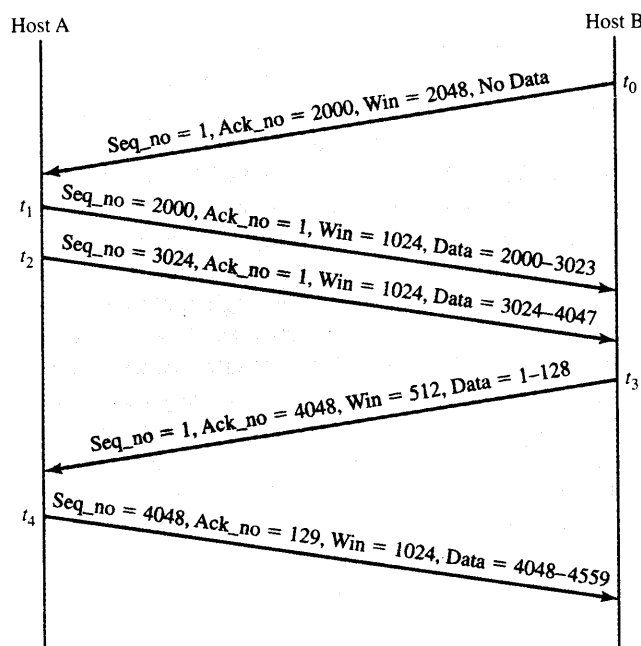


FIGURE 8.33 TCP window flow control.

at this time, host B finds out that it can allocate only 512 bytes of receive buffer space for this connection because other connections are also competing for the precious memory. So it shrinks the advertised window from 2048 bytes to 512 bytes. When host A receives the segment, the host changes its sending window to 512 bytes. If at time t_4 , host A has 2048 bytes of data to transmit then it will transmit only 512 bytes. We see that window advertisement dynamically controls the flow of data from the sender to the receiver and prevents the receiver's buffer from being overrun.

The previous discussion shows how TCP can delay transmission so that the acknowledgments can be piggybacked to the data segment. Another use of delayed transmission is to reduce bandwidth waste. (Consider a login session in which a user types one character at a time. When a character arrives from the application, the TCP module sends the segment with one byte of data to the other end. The other end (login server) needs to send an acknowledgment and then an echo character back to the client. Finally, the client needs to send an acknowledgment of the echo character. Thus one character generates four exchanges of IP packets between the client and the server with the following lengths: 41 bytes, 40 bytes, 41 bytes, and 40 bytes (assuming IP and TCP header are 20 bytes each). In the WAN environment, this waste of bandwidth is usually not justified.)

A solution to reduce the waste was proposed by Nagle and is called the **Nagle algorithm**. The idea works as follows. When an interactive application wants to send a character, the TCP module transmits the data and waits for the acknowledgment from the receiver. In the meantime, if the application generates more characters before the acknowledgment arrives, TCP will not transmit the characters but buffer them instead. When the acknowledgment eventually arrives, TCP transmits all the characters that have been waiting in the buffer in a single segment.)

In the LAN environment where delay is relatively small and bandwidth is plentiful, the acknowledgment usually comes back before another character arrives from the application. Thus the Nagle algorithm is essentially disabled. In the WAN environment where acknowledgments can be delayed unpredictably, the algorithm is self-adjusting. When delay is small, implying that the network is lightly loaded, only a few characters are buffered before an acknowledgment arrives. In this case TCP has the luxury of transmitting short segments. However, when delay is high, indicating that the network is congested, many more characters will be buffered. Here, TCP has to transmit longer segments and less frequently. In some cases the Nagle algorithm needs to be disabled to ensure the interactivity of an application even at the cost of transmission efficiency.

Another problem that wastes network bandwidth occurs when the sender has a large volume of data to transmit and the receiver can only deplete its receive buffer a few bytes at a time. Sooner or later the receive buffer becomes full. When the receiving application reads a few bytes from the receive buffer, the receiving TCP sends a small advertisement window to the sender, which quickly transmits a small segment and fills the receive buffer again. This process goes on and on with many small segments being transmitted by the sender for a single application message. This problem is called the **silly window syndrome**. It can be avoided by having the receiver not advertise the window until the window size is at least as large as half of the receive buffer size, or the maximum segment size. The sender side can cooperate by refraining from transmitting small segments.