Table 7.1 Use of Dijkstra's algorithm

k	β_{AC}	β_{AF}	β_{AE}	β_{AD}	β_{AG}	β_{AB}
{A}	AC(1)	AF(1)	×	×	×	×
{A,C}	AC(1)	AF(1)	ACE(3)	ACD(6)	×	×
{A,C,F}	AC(1)	AF(1)	ACE(3)	ACD(6)	AFG(7)	×
{A,C,F,E}	AC(1)	AF(1)	ACE(3)	ACED(4)	AFG(7)	×
{A,C,F,E,D}	AC(1)	AF(1)	ACE(3)	ACED(4)	ACEDG(6)	ACEDB(7)
{A,C,F,E,D,G}	AC(1)	AF(1)	ACE(3)	ACED(4)	ACEDG(6)	ACEDB(7)
{A,C,F,E,D,G,B}	AC(1)	AF(1)	ACE(3)	ACED(4)	ACEDG(6)	ACEDB(7)

2. **Initialize:**
   $\beta_{sj}(0) = \times$, for all $j \notin s$
   $\beta_{ss}(\ell) = 0$, for all ℓ
3. **Least-cost path:**
   for any node $j \neq s$ with predecessor node i:
   $\beta_{sj}(\ell + 1) = \min_i\{\beta_{si}(\ell) + \alpha_{ij}\}$ ■

If any two nodes i and j are not connected directly, $\beta_{ij}(\ell) = \times$. At step 2, every value of β is initialized. At step 3, we increase the number of links ℓ in a sequence of iterations. During each iteration, we find the least-cost path, given the value of ℓ. The algorithm ends when all nodes have been visited and included in the algorithm.

Example. Use the Bellman-Ford algorithm to find the least-cost path from node A to node B in Figure 7.2.

Solution. Table 7.2 shows the details of least-cost-path iterations. For iteration $\ell = 1$, only AC with cost 1 and AF with cost 1 exist, owing to the restriction enforced by $\ell = 1$. This trend changes at iteration $\ell = 2$, when ACE with cost 3 can be added to the set of least-cost paths. As seen, the result of the final least-cost path is identical to the one obtained by Dijkstra's algorithm.

We can now compare these two algorithms. In step 2 of Bellman-Ford, the calculation of the link cost to any node j requires knowledge of the link cost to all neighboring nodes. In step 3 of Dijkstra's algorithm, each node requires knowledge of the network topology at each time of iteration. The performance of each algorithm varies network to network and depends on the topology and size of a particular network. The comparison of these two algorithms, therefore, depends on the speed of each to achieve its

Table 7.2 Use of the Bellman-Ford algorithm

l	β_{AC}	β_{AF}	β_{AE}	β_{AD}	β_{AG}	β_{AB}
0	×	×	×	×	×	×
1	AC(1)	AF(1)	×	×	×	×
2	AC(1)	AF(1)	ACE(3)	ACD(6)	AFG(7)	×
3	AC(1)	AF(1)	ACE(3)	AED(4)	AFG(7)	ACDB(9)
4	AC(1)	AF(1)	ACE(3)	AED(4)	ACEDG(6)	ACEDB(7)

objective at its corresponding step given a network under routing. A simulation can clarify the efficiency of each algorithm given a certain network topology.

7.3 Non-Least-Cost-Path Routing

Our networking infrastructure deploys a variety of procedures, algorithms, and protocols for routing packets, depending on the applications, their significance, and the budget for building a network. Besides the least-cost-path algorithms that effectively identify the best possible paths, some nonoptimal algorithms can be used for applications that may not need complex and expensive routing protocols. Two examples of such *non-least-cost routing algorithms* are *flood routing* and *deflection routing*.

7.3.1 Flood Routing

Flood routing is a very simple routing strategy involving less hardware setup. The essence of this routing method is that a packet received from a node is copied and transmitted on all outgoing links of that node except for the link that the packet arrived from. After the first transmission, all the routers within one hops receive the packet. After the second transmission, all the routers within two hop receive the packet, and so on. Unless a mechanism stops the transmission, the process continues; as a result, the volume of traffic increases with time, as shown in Figure 7.3.

In this figure, three packets arrive at node A from a source. The first packet is copied to both nodes B and C. At nodes B and C, the copies of the packet are copied to their neighboring nodes. This method has the deficiency of packet reflection: a node can receive an unwanted copy of a packet. Although this problem can be fixed by dropping unwanted packets at any time, the network does not function effectively, owing to increased unnecessary traffic. one way to prevent packet duplication is to set

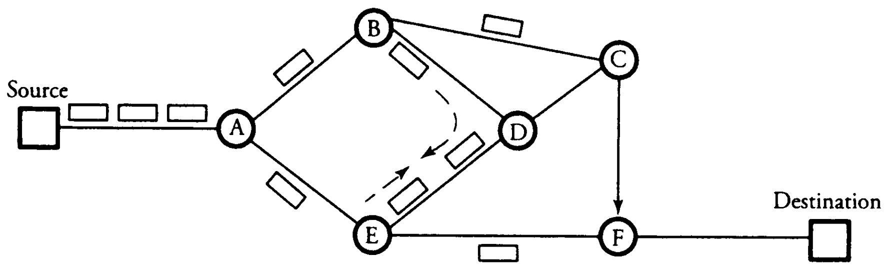

Figure 7.3 Flood routing

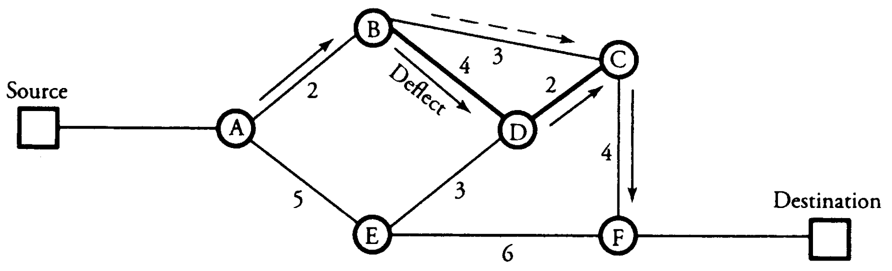

Figure 7.4 Deflection routing

up memory for each node to remember the identity of those packets that have already been retransmitted for blocking them.

7.3.2 Deflection Routing

In *deflection routing*, or *hot-potato routing*, a packet is sent to a destination determined at each router. At each step of the routing, a packet is examined with respect to its destination. If the requested link is free, the packet is sent on that link; otherwise, the packet is *deflected* onto another link, selected at random. The deflected packet is given an increment on its priority field. This increment gives the packet a better chance to win the contention with others in future contentions. For the deflected packet, if a new contention occurs with another packet in its next hop, the packet with the higher priority gets the desired link, and the other one is deflected with, of course, an increment in its priority field. In Figure 7.4, a packet is deflected at node B but eventually gets to node C, with a total of one additional hop and a total cost difference of 3, using node B.

7.4 Intradomain Routing Protocols

There are two classes of routing protocols: *intradomain routing protocol* or *intranetwork routing protocol* or *intranet*, and *interdomain routing protocol* or *internetwork routing protocol* or *extranet*. An intradomain routing protocol routes packets within a defined domain, such as for routing e-mail or Web browsing within an institutional network. By contrast, an interdomain routing protocol is a procedure for routing packets on networks of domains and is discussed in Section 7.5.

Figure 7.5 shows intradomain routing; each point-to-point link connects an associated pair of routers and indicates the corresponding cost of the connection. A *host* is an end system that can be directly connected to the router. A *cost* is considered with the output side of each router interface. When nothing is written on an arrow, no cost is associated with that connection. An arrow that connects a network and a router has a zero cost. A database is gathered from each router, including costs of paths in all separate directions. In the figure, two hosts within a domain are exchanging data facing their own LANs (N1 and N6), several other LANs (N2 through N5), and several routers (R1 through R8).

The most widely used intranetworking routing protocols are the two unicast routing protocols RIP and OSPF.

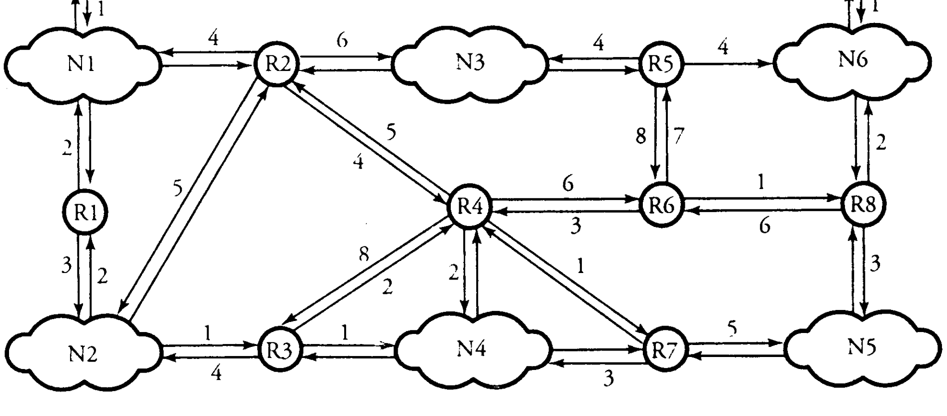

Figure 7.5 Intradomain routing, with two hosts in a large domain exchanging data facing several LANs and routers

Table 7.3 A routing table in router R1 in Figure 7.5a, using RIP

Destination Network	Next Router	Path	Cost
N1	—	R1-N1	2
N2	—	R1-N2	3
N3	R2	R1-N2-R2-N3	8
N4	R3	R1-N2-R3-N4	4
N5	R3	R1-N2-R3-N4-R7-N5	9
N6	R2	R1-N2-R3-N4-R7-N5-R8-N6	11

7.4.1 Routing Information Protocol (RIP)

The *routing information protocol* (RIP) is a simple intradomain routing protocol. RIP is one of the most widely used routing protocols in the Internet infrastructure but is also appropriate for routing in smaller domains. In RIP, routers exchange information about reachable networks and the number of hops and associated costs required to reach a destination. This protocol is summarized as follows.

Begin RIP Algorithm

1. Apply the Bellman-Ford algorithm in a distributed fashion, including all hosts and routers.
2. For each router, form an optimum vector of distance indicating the cost of routing and other necessary parameters, using *distance-vector routing* (discussed next).
3. If a cost of routing at any point changes, propagate the change periodically to the neighboring routers and hosts of that point.
4. Update tables of routers and hosts periodically. ■

Table 7.3 shows routing information maintained in router R1 of the network shown in Figure 7.5. For example, at the third row of the routing table, the path to N3 is initiated through router R2 and established through path R1-N2-R2-N3, with a total cost of 8. Because routers and hosts within the network use vectors of data, a technique called *distance vector routing* is used.

Distance Vector Routing

The *distance vector algorithm* was designed mainly for small network topologies. The term *distance vector* derives from the fact that the protocol includes its routing updates with a vector of distances, or hop counts. In distance vector routing, all nodes exchange information only with their neighboring nodes. Nodes participating in the same local network are considered neighboring nodes. In this protocol, each individual node i maintains three vectors:

$\mathbf{B}_i = [b_{i,1}, \ldots, b_{i,n}] =$ *link-cost vector*

$\mathbf{D}_i = [d_{i,1}, \ldots, d_{i,m}] =$ *distance vector*

$\mathbf{H}_i = [h_{i,1}, \ldots, h_{i,m}] =$ *next-hop vector*

For the calculation of the link-cost vector, we consider node i to be directly connected to n networks out of the total of m existing networks. The link-cost vector $\mathbf{B}_i$ is a vector containing the costs of node i to its directly connected network. For example, $b_{i,1}$ refers to the cost of node i to network 1. Distance vector $\mathbf{D}_i$ contains the estimated minimum delays from node i to all networks within its domain. For example, $d_{i,1}$ is the estimated minimum delay from node i to network 1. Finally, the next-hop vector $\mathbf{H}_i$ is a matrix vector containing the next node in the minimum-delay path from node i to its directly connected network. As an example for this vector, $h_{i,1}$ refers to the next node in the minimum-delay path from node i to Network 1. Each node exchanges its distance every r seconds with all its neighbors. Node i is responsible for updating both of its vectors by:

$$d_{i,j} = \min_{k \in \{1, \cdots, n\}} \left[d_{k,j} + b_{i,g(x,y)} \right] \tag{7.1}$$

and

$$h_{i,j} = k, \tag{7.2}$$

where $\{1, \ldots, n\}$ is the set of network nodes for node i, and $g(x, y)$ is a network that connects node x to node y; thus, $b_{i,g(x,y)}$ is the cost of node i to network $g(x, y)$. Equations (7.1) and (7.2) are a distributed version of the Bellman-Ford algorithm, which is used in RIP. Each router i begins with $d_{i,j} = b_{i,j}$ if it is directly connected to network j. All routers concurrently replace the distance vectors and compute Equation (7.1). Routers repeat this process again. Each iteration is equal to one iteration of step 2 in the Bellman-Ford algorithm, processed in parallel at each node of the graph. The shortest-length paths with a distance of at most one hop are considered after the first iteration. Then, the second iteration takes place, with the least-cost paths with at most two hops; this continues until all least-cost paths have been discovered.

Updating Routing Tables in RIP

Since RIP depends on distance vector routing, each router transmits its distance-vector to its neighbors. Normally, updates are sent as a replies whether or not requested. When a router broadcasts an RIP request packet, each router in the corresponding domain receiving the request immediately transmits a reply. A node within a short window of time receives distance-vectors from all its neighbors, and the total update occurs, based on incoming vectors. But this process is not practical, since the algorithm is asynchronous, implying that updates may not be received within any specified window of time.

RIP packets are sent typically in UDP fashion, so packet loss is always possible. In such environments, RIP is used to update routing tables after processing each distance vector. To update a routing table, if an incoming distance vector contains a new destination network, this information is added to the routing table. A node receiving a route with a smaller delay immediately replaces the previous route. Under certain conditions, such as after a router reset, the router can receive all the entries of its next hop to reproduce its new table.

The RIP *split-horizon* rule asserts that it is not practical to send information about a route back in the direction from which it was received. The split-horizon advantage is increased speed and removal of incorrect route within a timeout. The RIP *poisoned-reverse* rule has a faster response and bigger message size. Despite the original split horizon, a node sends updates to neighbors with a hop count of 16 for routing information arrived from those neighbors.

RIP Packet Format

Figure 7.6 shows the packet format of an RIP header. Each packet consists of several address distances. The header format contains the following specifications for the first address distance.

- *Command* indicates a request with value 1 or a reply with value 2.
- *Version number* specifies the version: RIP-1 or RIP-2.
- *Address family identifier* shows the type of address, such as an IP address.
- *IP address* provides the IP address in a particular network.
- *Metric* identifies the distance from a router to a specified network.

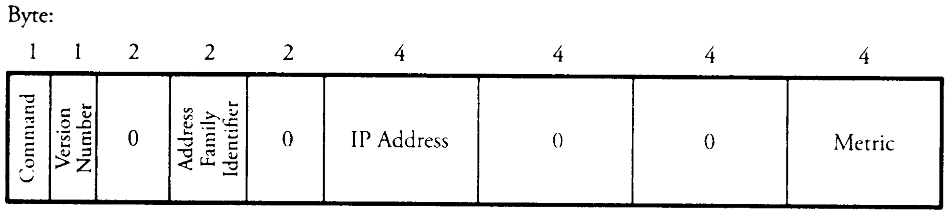

Figure 7.6 Routing Information Protocol (RIP) header

Table 7.4 Current routing table with updated values for Host 1 in Figure 7.5, using RIP

Destination Network	Next Router	Cost	Updated Next Router	Updated Cost
N1	—	1	—	1
N2	R1	4	*R2*	*2*
N3	R2	7	R2	7
N4	R1	8	R1	8
N5	R2	11	R2	11
N6	R2	13	R2	*9*

If a link cost is set to 1, the *metric* field acts as a hop count. But if link costs have larger values, the number of hops becomes smaller. Each packet consists of several address distances. If more than one address distance is needed, the header shown in Figure 7.6 can be extended to as many address distances as are requested, except for the *command* and *version number* and its 2 bytes of 0s; the remaining field structure can be repeated for each address distance.

Example. Apply RIP for connecting Host 1 to all six networks in Figure 7.5, given that at a certain time, the cost of R2-N2 changes from 5 to 1, and the cost of R4-R6 changes from 6 to 2.

Solution. The current routing table for the host is shown in Table 7.4. In particular, note the status of the table on destination network N2, where the next router is R1 with path Host1-N1-R1-N2 and a cost of 4. Changes on the link costs affect the status of the N2 and N6 entries. The new updates are reflected in Table 7.4, where the new path becomes Host1-R2-N2, with a new, lower cost of 2. The status of N6 also changes to a new path, Host1-R2-R4-R6-R8-N6, with a new, lower cost of 9.

Issues and Limitations of RIP

One of the disadvantages of RIP is its slow convergence in response to a change in topology. *Convergence* refers to the point in time at which the entire network becomes updated. In large networks, a routing table exchanged between routers becomes very large and difficult to maintain, which may lead to an even slower convergence. Also, RIP might lead to suboptimal routes, since its decision is based on hop counts. Thus, low-speed links are treated equally or sometimes preferred over high-speed links. Another issue with RIP is the *count-to-infinity* restriction. Distance vector protocols have a limit on the number of hops after which a route is considered inaccessible. This restriction would cause issues for large networks.

Other issues with RIP stem from the distance vector algorithm. First, reliance on hop counts is one deficiency, as is the fact that routers exchange all the network values via periodic broadcasts of the entire routing table. Also, the distance vector algorithm may cause loops and delays, since they are based on periodic updates. For example, a route may go into a *hold* state if the corresponding update information is not received in a certain amount of time. This situation can translate into a significant amount of delay in update convergence in the network before the network discovers that route information has been lost.

Another major deficiency is the lack of support for variable-length subnet masks. RIP does not exchange mask information when it sends routing updates. A router receiving a routing update uses its locally defined subnet mask for the update, which would lead to complications and misinterpretation in a variably subnetted network. Distance vector networks do not have hierarchies, which makes them incapable of integrating to larger networks.

7.4.2 Open Shortest Path First (OSPF)

The limitations of [illegible] nonpreferred routing strategy as networks scale up. The *Open Shortest Path First* (OSPF) protocol is a better choice of intradomain routing protocols, especially for TCP/IP applications. OSPF is based on Dijikstra's algorithm, using a tree that describes the network topology to define the shortest path from each router to each destination address. Since it keeps track of all paths of a route, OSPF has more overhead than RIP but provides more stability and useful options. The essence of this protocol is as follows.

Begin OSPF Protocol

1. Apply Dijikstra's algorithm or another efficient algorithm in a distributed fashion for each host and router.
2. In each host or router, calculate the least-cost path, and propagate it to all nodes, using *link-state routing.*
3. Periodically propagate any changes in routing cost to all routers and hosts.
4. Update the routing tables of routers and hosts. ■

Before focusing on the details of OSPF, we need to describe the essence of the *link-state routing* protocol.

Link-State Routing

In the *link-state routing*, routers collaborate by exchanging packets carrying status information about their adjacent links. A router collects all the packets and determines the network topology, thereby executing its own shortest-route algorithm.

As explained in the section on RIP, with distance vector routing, each router must send a distance vector to all its neighbors. If a link cost changes, it may take a significant amount of time to spread the change throughout the network. *Link-state routing* is designed to resolve this issue; each router sends routing information to all routers, not only to the neighbors. That way, the transmitting router discovers link-cost changes, so a new link cost is formed. Since each router receives all link costs from all routers, it is able to calculate the least-cost path to each destination of the network. The router can use any efficient routing algorithm, such as Dijkstra's algorithm, to find the shortest path.

The core function of the link-state algorithm is flood routing, which requires no network topology information. Let's review the three important properties of flooding used in link-state routing. First, a packet always traverses completely between a source and a destination, given at least one path between these two nodes. This property makes this routing technique robust. Second, at least one copy of the packet arriving at the destination must possess the minimum delay, since all routers in the network are tried. This property makes the flooded information propagate to all routers very quickly. Third, all nodes in the network are visited whether they are directly or indirectly connected to the source node. This property makes every router receive all the information needed to update its routing table.

Details of OSPF Operation

To return to the OSPF operation: Every router using OSPF is aware of its local link-cost status and periodically sends updates to all routers. After receiving update packets, each

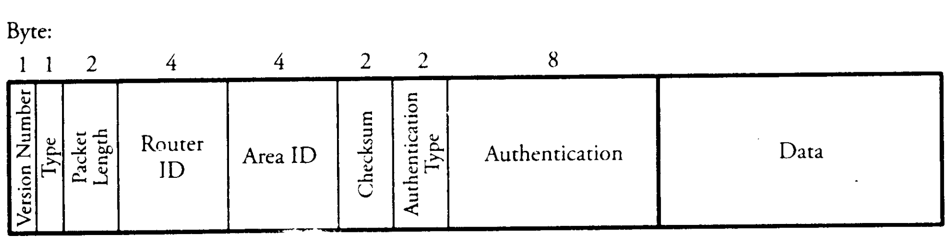

Figure 7.7 OSPF header format

router is responsible for informing its sending router of receipt of the update. These communications, though, result in additional traffic, potentially leading to congestion. OSPF can provide a flexible link-cost rule based of type of service (TOS). The TOS information allows OSPF to select different routes for IP packets, based on the value of the TOS field. Therefore, instead of assigning a cost to a link, it is possible to assign different values of cost to a link, depending on the TOS the value for each block of data.

TOS has five levels of values: *level 1* TOS, with the highest value to act by default, through *level 5* TOS, with the lowest value for minimized delay. By looking at these five categories, it is obvious that a router can build up to five routing tables for each TOS. For example, if a link looks good for delay-sensitive traffic, a router gives it a low, level 5 TOS value for low delay and a high, level 2 TOS value for all other traffic types. Thus, OSPF selects a different shortest path if the incoming packet is normal and does not require an optimal route.

OSPF Packet Format

An IP packet that contains OSPF has a standard broadcast IP address of 224.0.0.5 for flood routing. All OSPF packets use a 24-byte header as follows:

- *Version number* indicates the version of OSPF.
- *Type* is one of the five types of packets for OSPF to choose from: *hello*, *database description*, *link-state request*, *link-state update*, and *link-state acknowledgment*.
- *Packet length* specifies the length of the OSPF packet.
- *Router* ID specifies the packet's source router ID.
- *Area* ID refers to the area that the source router belongs to.
- *Checksum* specifies the standard IP checksum of the packet contents (see Chapter 4).

- *Authentication type* identifies which authentication method to choose (see Section 10.5).
- *Authentication* specifies the authentication method (see Section 10.5).

The *hello* packet specified in the *Type* field is used to detect each router's active neighbors. Each router periodically sends out *hello* packets to its neighbors to discover its active neighboring routers. Each packet contains the identity of the neighboring router interface taken from the *hello* packet already received from it. The *database description* packets are used for database structure exchange between two adjacent routers to synchronize their knowledge of network topology. The *link-state request* packet is transmitted to request a specific portion of link-state database from a neighboring router. The *link-state update* packet transfers the link-state information to all neighboring routers. Finally, the *link-state acknowledgment* packet acknowledges the update of a link-state packet.

In summary, when a router is turned on, it transmits *hello* packets to all neighboring routers and then establishes routing connections by synchronizing databases. Periodically, each router sends a link-state update message describing its routing database to all other routers. Therefore, all routers have the same description of the topology of the local network. Every router calculates a shortest-path tree that describes the shortest path to each destination address, thereby indicating the closest router for communications.

7.5 Interdomain Routing Protocols

Unlike intradomain routing protocols, *interdomain routing protocols* create a network of networks, or an *internetwork*. Interdomain routing protocols route packets outside of a defined domain. Each domain consists of several networks and routers that can be accessed publicly. Figure 7.8 shows an example of internetworking, in which a packet at router R_1 faces two defined larger service provider domains, A and B. Each service domain is shown by a number of interconnected circles, each representing a network or a router. Obviously, at router R_1, finding the best path for a packet to go to which domain is a challenge. The Border Gateway Protocol helps meet this challenge.

7.5.1 Border Gateway Protocol (BGP)

The *Border gateway protocol* (BGP) is a preferred routing protocol for interdomain communications and TCP/IP connections. BGP allows routers to carry specific policies or constraints that they must meet. With BGP, routers exchange more comprehensive information about routes to a certain destination instead of simply costs and the best

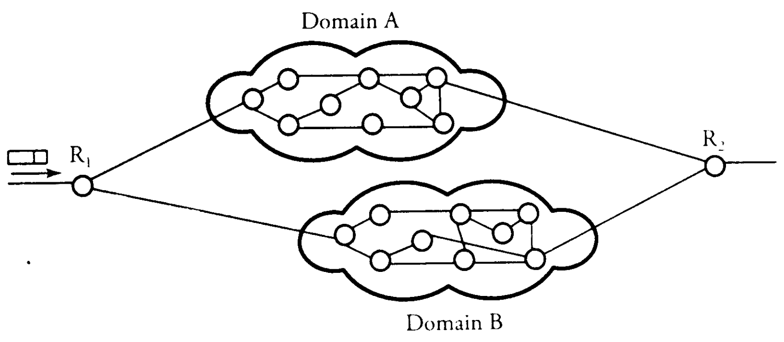

Figure 7.8 Internetworking, in which router packet faces two defined service domains

link. In BGP, two contributing routers can exchange routing information even if they are located in two different autonomous systems. When an external destination is chosen, a router sends the information to all internal neighbors. Then, all routers decide whether the new route is possible, and if so, the new route is added to the router's database. Thus, the new update message is propagated. One of the most important techniques in BGP is path-vector routing.

Path-Vector Routing

RIP and OSPF are not suitable for interdomain routing protocols. As discussed earlier, distance vector routing is used to send information to each of a router's neighbors, and then each router builds up a routing database. However, a router is not aware of the identity of routers on any particular path. Two problems arise. First, if different routers give different information to an assigned cost, it is impossible to have stable and loop-free routes. Second, an autonomous system can have limitations about which specific autonomous system should be used. This is true while the distance vector algorithm has no information about the autonomous systems.

Each router sends its link cost to all other routers and then starts routing calculations. Two issues can arise in link-state routing. First, different independent systems can use different costs and have different limitations. The link-state protocol allows a router to make the topology, and its metrics may be different for each independent system. In this case, it is impossible to create a reliable routing algorithm. Second, when flood routing occurs, the use of an interdomain routing protocol across the independent system can be unstable.

To resolve these issues, consider an alternative solution: the *path vector routing protocol*, which provides information about how to reach a network given a certain router and identifies which autonomous system (or domain) should be visited, as in the case for router R_1 in Figure 7.8. The path vector routing protocol is different from the distance vector algorithm, in which each path has information about cost and distance. In the path vector routing protocol, these packages of information are not included, and all visited autonomous systems and all components of domain A in Figure 7.8 reaching the destination network are listed in each routing information package. Thus, a router can be programmed to refuse the acceptance of a particular path if the information about the path is not included in the package it receives.

Details of BGP

BGP was created to find a solution to interdomain routing among autonomous (independent) systems. BGP works well for making a connection when a long-haul TCP session must be established. BGP has three functional components:

1. Neighbor relationship
2. Neighbor maintenance
3. Network maintenance

The neighbor relationship refers to an agreement between two routers in two different autonomous systems to exchange routing information on a regular basis. A router may reject its participation in establishing a neighbor relationship for several reasons, such as the rule of the domain, overload, or a temporary malfunctioning of external links. Neighbor maintenance is a process of maintaining the neighbor relationship already established. Normally, each corresponding router needs to find out whether the relationship with the other router is still available. For this reason, two routers send *keep-alive* messages to each other. The last BGP process is network maintenance. Each router keeps the database of the subnetworks that it can reach and tries to get the best route for that subnetwork.

BGP Packets

There are four different BGP packets as shown in Figure 7.9, as follows:

- *Open packet*. This packet requests establishment of a relationship between two routers.
- *Update packet*. This packet conveys update information about routes.

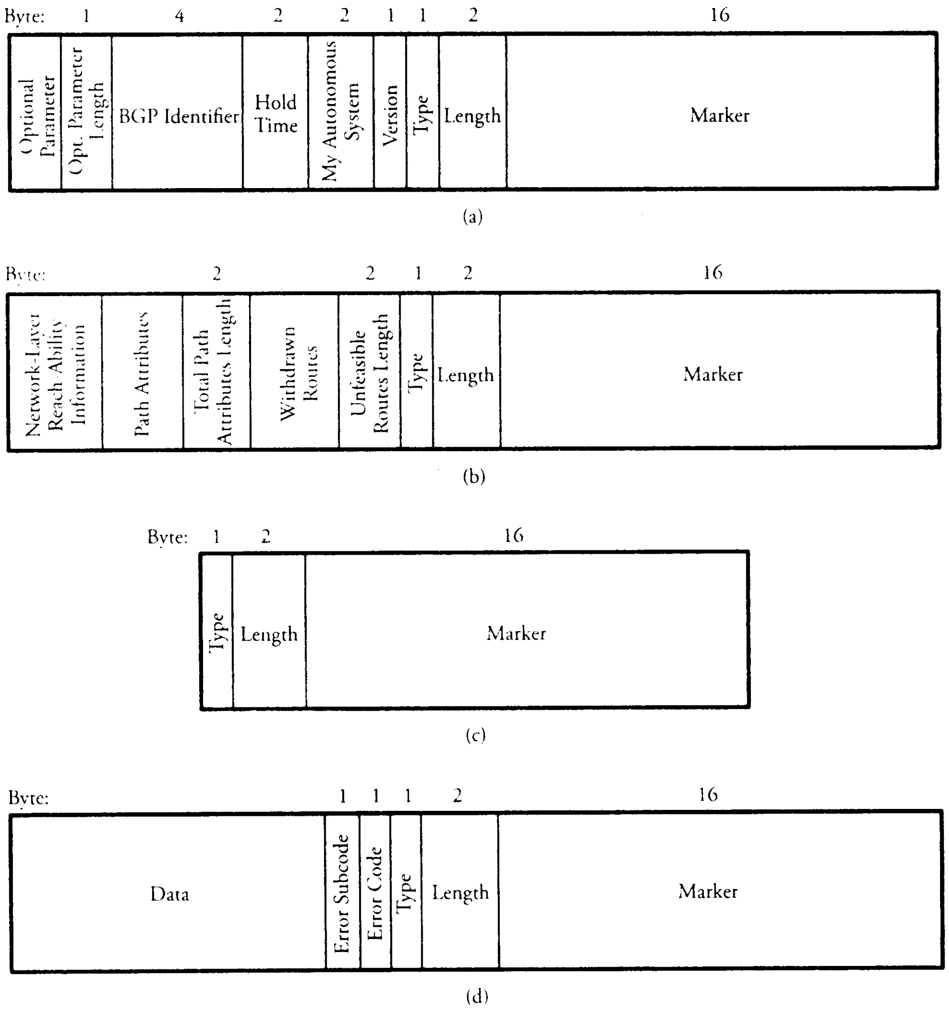

Figure 7.9 Four types of BGP packets and their fields: (a) open packet, (b) update packet, (c) keep-alive packet, and (d) notification packet

- *Keep-alive packet.* Once a relationship between two routers is established, this packet confirms its neighbor relationship frequently.
- *Notification packet.* This packet is used when an error occurs.

Figure 7.10 shows a BGP connection. Assume that router R_2 in one Internet service provider, network 2 opens a TCP connection to its desired router, R_3, in another ISP

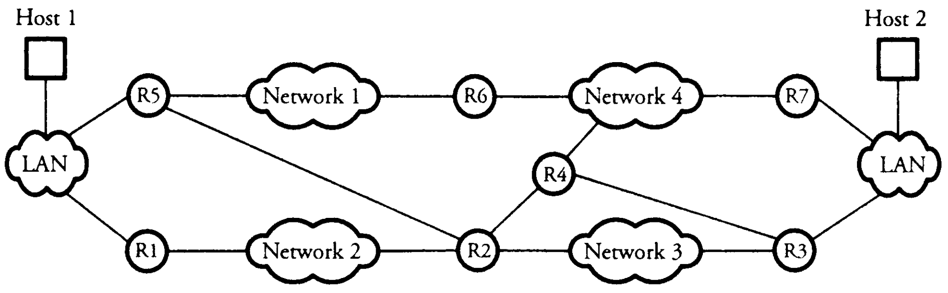

Figure 7.10 Use of BGP in an interdomain routing protocol

domain of network 3. Router R_2 sends an *open* packet to R_3. This packet is identified by R_3, telling it which domain the sender belongs to. A router sends a *keep-alive* packet to its neighbors to prevent an agreed hold time from expiring.

The first type of update packet can include the information of a single route that passes through a network. This information can be divided into three fields: the *network-layer readability* field, the *path attributes* field, and the *total path attributes length* field. *The network-layer readability* field has a list of subnetwork identifiers that can be found by the router. The *path attributes* field has a list of attributes that can be referred to a specific route. The second type of update information is used to remove one or more routes identified by the IP address of the destination subnetwork. In this case, the *notification* packet is sent in case of an error, which may be authentication errors, validity errors in the update packet, or an expired hold-time error.

7.6 Congestion Control at Network Layer

Congestion represents an overloaded condition in a network. Congestion control can be achieved by optimum usage of the available resources in the network. Figure 7.11 shows a set of performance graphs for networks in which three possible cases are compared: no congestion, moderate congestion, and severe congestion. These plots indicate that if a network has no congestion control, the consequence may be a severe performance degradation, even to the extent that the carried load starts to fall with increasing offered load. The ideal situation is the one with no loss of data, as a result of no congestion, as shown in the figure. Normally, a significant amount of engineering effort is required to design a network with no congestion.

Congestion can be either *logical* or *physical*, as shown in Figure 7.12. The queueing feature in two routers can create a logical bottleneck between user A and user B. Meanwhile, insufficient bandwidth—a resource shortage—on physical links between

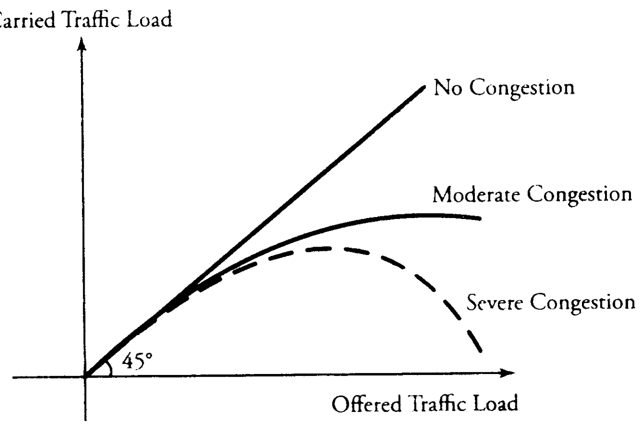

Figure 7.11 Comparison among networks in which congestion, moderate congestion, and severe congestion exist

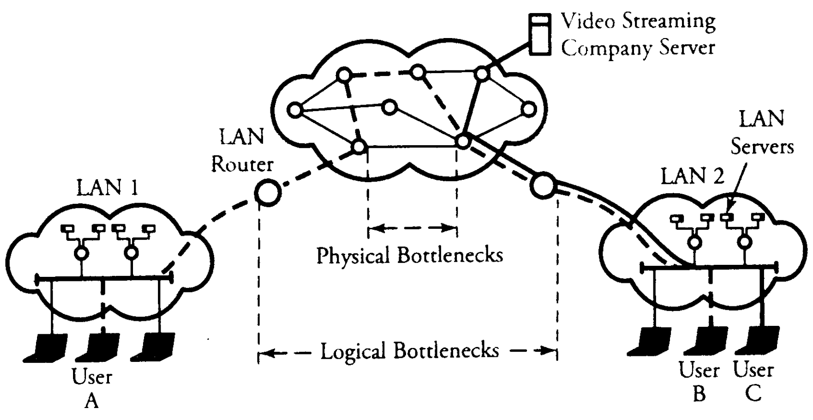

Figure 7.12 Forms of bottleneck along a round-trip path between two users

routers and the network can also be a bottleneck, resulting in congestion. Resource shortage can occur

- At the *link layer*, where the link bandwidth runs out
- At the *network layer*, where the queues of packets at nodes go out of control
- At the *transport layer*, where logical links between two routers within a communication session go out of control

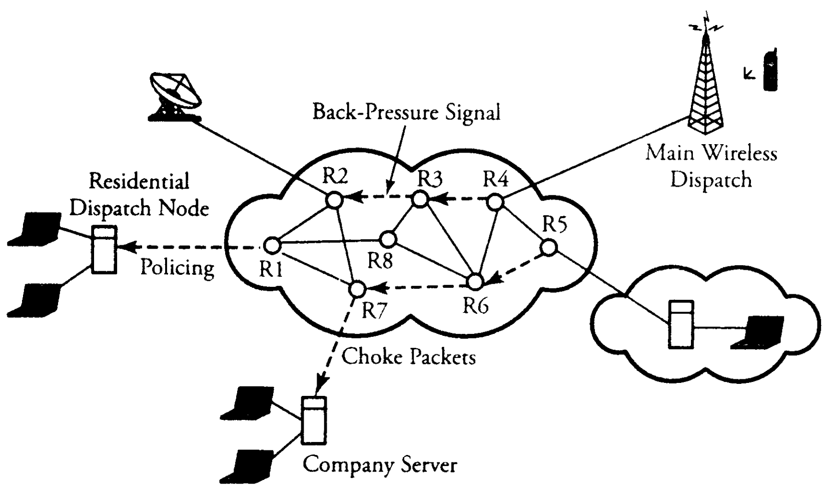

Figure 7.13 Unidirectional congestion control

One key way to avoid congestion is to carefully allocate network resources to users and applications. Network resources, such as bandwidth and buffer space, can be allocated to competing applications. Devising optimum and fair resource-allocation schemes can control congestion to some extent. In particular, congestion control applies to controlling data flow among a group of senders and receivers, whereas flow control is specifically related to the arrangement of traffic flows on links. Both resource allocation and congestion control are not limited to any single level of the protocol hierarchy. Resource allocation occurs at switches, routers, and end hosts. A router can send information on its available resources so that an end host can reserve resources at the router to use for various applications.

General methods of congestion control are either *unidirectional* or *bidirectional*. These two schemes are described next.

7.6.1 Unidirectional Congestion Control

A network can be controlled unidirectionally through *back-pressure signaling*, *transmission of choke packets*, and *traffic policing*. Figure 7.13 shows a network of eight routers: R1 through R8. These routers connect a variety of servicing companies: cellphones, satellite, residential, and company LANs. In such a configuration, congestion among routing nodes may occur at certain hours of the day.

The first type of congestion control is achieved by generating a *back-pressure signal* between two routers. The back-pressure scheme is similar to fluid flow in pipes. When one end of a pipe is closed, the pressure propagates backward to slow the flow of water at the source. The same concept can be applied to networks. When a node becomes congested, it slows down the traffic on its incoming links until the congestion is relieved. For example, in the figure, router R4 senses overloading traffic and consequently sends signals in the form of back-pressure packets to router R3 and thereby to router R2, which is assumed to be the source of overwhelming the path. Packet flow can be controlled on a hop-by-hop basis. Back-pressure signaling is propagated backward to each node along the path until the source node is reached. Accordingly, the source node restricts its packet flow, thereby reducing the congestion.

Choke-packet transmission is another solution to congestion. In this scheme, choke packets are sent to the source node by a congested node to restrict the flow of packets from the source node. A router or even an end host can send these packets when it is near full capacity, in anticipation of a condition leading to congestion at the router. The choke packets are sent periodically until congestion is relieved. On receipt of the choke packets, the source host reduces its traffic-generation rate until it stops receiving them.

The third method of congestion control is *policing* and is quite simple. An edge router, such as R1 in the figure, acts as a traffic police and directly monitors and controls its immediate connected consumers. In the figure, R1 is policing a residential dispatch node from which traffic from a certain residential area is flowing into the network.

7.6.2 Bidirectional Congestion Control

Figure 7.14 illustrates *bidirectional congestion control*, a host-based resource-allocation technique. The destination end host controls the rate at which it sends traffic, based on observable network conditions, such as delay and packet loss. If a source detects long delays and packet losses, it slows down its packet flow rate. All sources in the network adjust their packet-generation rate similarly; thus, congestion comes under control. Implicit signaling is widely used in packet-switched networks, such as the Internet.

In bidirectional signaling, network routing nodes alert the source of congested resources by setting bits in the header of a packet intended for the source. An alternative mechanism for signaling is to send control packets as choke packets to the source, alerting it of any congested resources in the network. The source then slows down its rate of packet flow. When it receives a packet, the source checks for the bit that indicates congestion. If the bit is set along the path of packet flow, the source slows down its

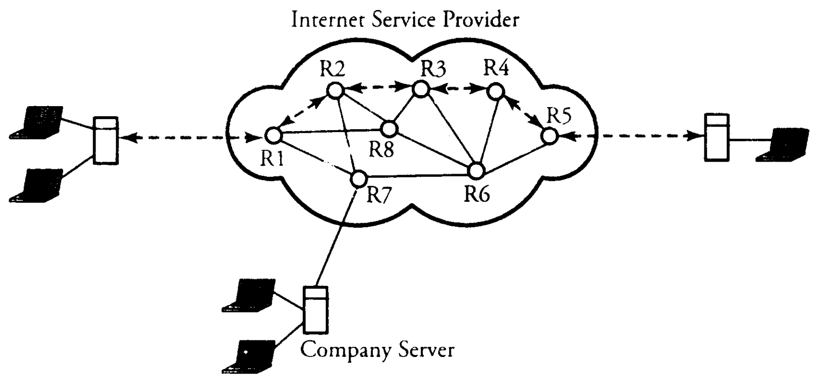

Figure 7.14 Bidirectional congestion control

sending rate. Bidirectional signaling can also be imposed by the window-based and rate-based congestion-control schemes discussed earlier.

7.6.3 Random Early Detection (RED)

Random early detection (RED) avoids congestion by detecting and taking appropriate measures early. When packet queues in a router's buffer experience congestion, they discard all incoming packets that could not be kept in the buffer. This *tail-drop policy* leads to two serious problems: global synchronization of TCP sessions and prolonged congestion in the network. RED overcomes the disadvantages of the tail-drop policy in queues by randomly dropping the packets when the average queue size exceeds a given minimum threshold.

From the statistical standpoint, when a queueing buffer is full, the policy of random packet drop is better than multiple-packet drop at once. RED works as a feedback mechanism to inform TCP sessions that the source anticipates congestion and must reduce its transmission rate. The packet-drop probability is calculated based on the weight allocation on its flow. For example, heavy flows experience a larger number of dropped packets. The average queue size is computed, using an exponentially weighted moving average so that RED does not react to spontaneous transitions caused by bursty Internet traffic. When the average queue size exceeds the maximum threshold, all further incoming packets are discarded.

RED Setup at Routers

With RED, a router continually monitors its own queue length and available buffer space. When the buffer space begins to fill up and the router detects the possibility of congestion, it notifies the source implicitly by dropping a few packets from the source. The source detects this through a time-out period or a duplicate ACK. Consequently, the router drops packets earlier than it has to and thus implicitly notifies the source to reduce its congestion window size.

The "random" part of this method suggests that the router drops an arriving packet with some drop probability when the queue length exceeds a threshold. This scheme computes the average queue length, $E[N_q]$, recursively by

$$E[N_q] = (1-\alpha)E[N_q] + \alpha N_i, \tag{7.3}$$

where N_i is the instantaneous queue length, and $0 < \alpha < 1$ is the weight factor. The average queue length is used as a measure of load. The Internet has bursty traffic, and the instantaneous queue length may not be an accurate measure of the queue length. RED sets minimum and maximum thresholds on the queue length, N_{min} and N_{max}, respectively. A router applies the following scheme for deciding whether to service or drop a new packet. If $E[N_q] \geq N_{max}$, any new arriving packet is dropped. If $E[N_q] \leq N_{min}$, the packet is queued. If $N_{min} < E[N_q] < N_{max}$, the arriving packet is dropped with probability P given by

$$P = \frac{\delta}{1 - c\delta}, \tag{7.4}$$

where coefficient c is set by the router to determine how quickly it wants to reach a desired P. In fact, c can be thought of as the number of arriving packets that have been queued. We can then obtain δ from

$$\delta = \frac{E[N_q] - N_{min}}{N_{max} - N_{min}}. \tag{7.5}$$

In essence, when the queue length is below the minimum threshold, the packet is admitted into the queue. Figure 7.15 shows the variable setup in RED congestion avoidance. When the queue length is between the two thresholds, the packet-drop probability increases as the queue length increases. When the queue length is above the maximum threshold, the packet is always dropped. Also, shown in Equation (7.4), the packet-drop probability depends on a variable that represents the number of arriving packets from a flow that has been queued. When the queue length increases, all that is needed is to drop one packet from the source. The source then halves its congestion window size.

Once a small number of packets are dropped, the associated sources reduce their congestion windows if the average queue length exceeds the minimum threshold, and

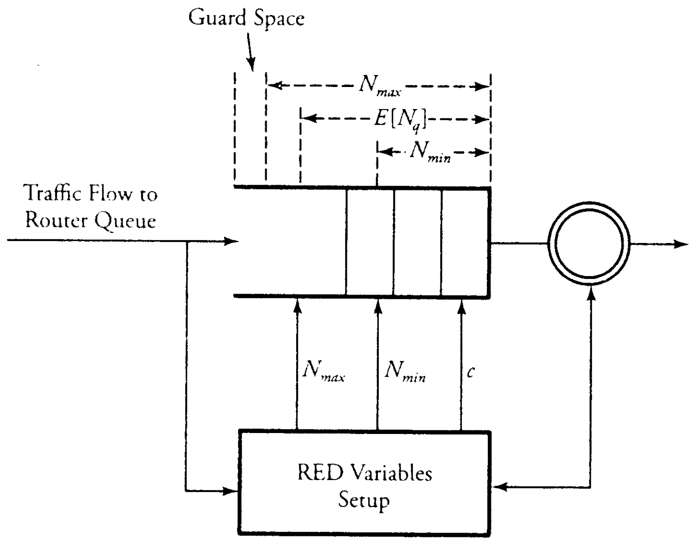

Figure 7.15 Variables setup in the RED congestion-avoidance method

therefore the traffic to the router drops. With this method, any congestion is avoided by an early dropping of packets.

RED also has a certain amount of fairness associated with it; for larger flows, more packets are dropped, since P for these flows could become large. One of the challenges in RED is to set the optimum values for N_{min}, N_{max}, and c. Typically, N_{min} has to be set large enough to keep the throughput at a reasonably high level but low enough to avoid congestion. In practice, for most networks on the Internet, the N_{max} is set to twice the minimum threshold value. Also, as shown by *guard space* in Figure 7.15, there has to be enough buffer space beyond N_{max}, as Internet traffic is bursty.

7.6.4 A Quick Estimation of Link Blocking

A number of techniques can be used to evaluate a communication network's blocking probabilities. These techniques can vary according to accuracy and to network architectures. One of the most interesting and relatively simple approaches to calculating the level of blocking involves the use of Lee's probability graphs. Although Lee's technique requires approximations, it nonetheless provides reasonably accurate results.

Serial and Parallel Connection Rules

Lee's method is based on two fundamental rules of *serial* and *parallel* connections. Each link is represented by its blocking probability, and the entire network of links

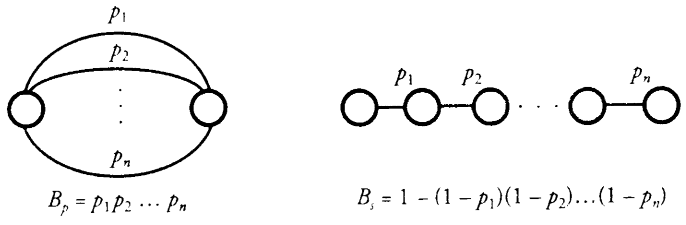
p_1
p_2
p_n
$B_p = p_1 p_2 \cdots p_n$
p_1
p_2
p_n
$B_s = 1 - (1 - p_1)(1 - p_2)\ldots(1 - p_n)$

Figure 7.16 Models for serial and parallel connection rules

is evaluated, based on blocking probabilities represented on each link, using one or both of the rules. This approach is easy to formulate, and the formula directly relates to the underlying network structures, without requiring any other detailed parameters. Thus, Lee's approach provides insight into the network structures, giving a remarkable solution for performance evaluations.

Let p be the probability that a link is busy, or the percentage of link utilization. Thus, the probability that a link is idle is denoted by $q = 1 - p$. Now, consider a simple case of two links in parallel to complete a connection with probabilities p_1 and p_2. The composite blocking probability, B_p, is the probability that both links are in use, or

$$B_p = p_1 p_2. \tag{7.6}$$

If these two links are in series to complete a connection, the blocking probability, B_s, is determined as 1 minus the probability that both links are available:

$$B_s = 1 - (1 - p_1)(1 - p_2). \tag{7.7}$$

We can generalize the estimation of a network of links by two fundamental rules.

1. Rule 1: For a parallel connection of links, the blocking probability is estimated by forming the product of the blocking probabilities for the subnetworks, as shown in Figure 7.16. Let a source and a destination be connected in general through n links with probabilities p_1 through p_n, respectively, as shown in Figure 7.16. Therefore, if the source and the destination are linked through parallel connections, the probability of blocking B_p is obtained from a product form as follows:

$$B_p = p_1 p_2 \ldots p_n. \tag{7.8}$$

2. Rule 2: For a serial connection of links, the probability of blocking is estimated by forming the product of the probabilities of no blocking for the network. This method makes the assumption that the probability that a given link is busy is

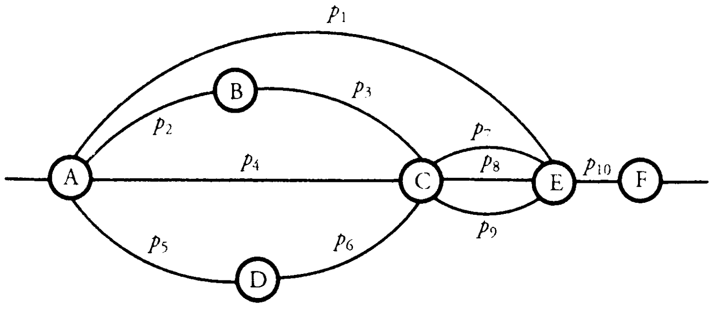

Figure 7.17 A network of links and their blocking probabilities

independent from link to link. Although this independence assumption is not strictly correct, the resulting estimates are sufficiently accurate. If links are in series, the probability of blocking is obtained from

$$B_s = 1 - (1 - p_1)(1 - p_2)...(1 - p_n). \tag{7.9}$$

Example. The network in Figure 7.17 has six nodes, A through F, interconnected with ten links. The corresponding blocking probabilities of links, p_1 through p_{10}, are indicated on the corresponding links. For this network, find the overall blocking probability from A to F.

Solution. This network consists of a number of serial and parallel connections: for example, $p_{ADC} = 1 - (1 - p_5)(1 - p_6)$, $p_{ABC} = 1 - (1 - p_2)(1 - p_3)$, and $p_{CE} = p_7p_8p_9$. Thus:

$$B = p_{AF} = 1 - \left\{1 - \left[1 - (1 - p_4.p_{ABC}.p_{ADC})(1 - p_{CE})\right] p_1\right\} (1 - p_{10}).$$

7.7 Summary

This chapter focused on layer 3 of the protocol stack reference model. Two least-cost-path algorithms are Dijkstra's algorithm—a centralized least-cost algorithm designed to find the best path from a source to a destination through the optimization of path costs—and the Bellman-Ford algorithm—a distributed least-cost approach by which each node retrieves information from reachable nodes to each of its neighbors. Non-least-cost algorithms—*deflection*, or flood algorithms—are not optimal but they have their own useful applications.

A number of practical routing protocols exist. One category is intradomain routing protocols, such as OSPF and RIP. A router equipped with OSPF is aware of its local link-cost status and periodically sends updates to all surrounding routers. OSPF is based on *link-state routing*, by which routers exchange packets carrying status information of their adjacent links. A router collects all the packets and determines the network topology, thereby executing its own shortest-route algorithm. RIP is one of the most widely used routing protocols in the Internet infrastructure; routers exchange information abour reachable networks and the number of hops. RIP uses the Bellman-Ford algorithm.

By contrast, *interdomain routing protocols*, such as BGP, let two contributing routers exchange routing information even if they are located in two different autonomous systems. Each router sends information to all internal neighbors and decides whether the new route is possible.

Congestion control at the network layer is a major issue. In computer networks, congestion represents some form of overload, generally resulting from resources shortage at links and devices. *Unidirectional congestion control* can take several forms: ***back-pressure signaling***, *transmission of choke packets*, and *traffic policing*. ***Bidirectional congestion control*** is a host-based resource-allocation scheme. In this category, ***random early detection*** is one congestion-avoidance technique. An approximation method to calculate ***link blocking*** follows two simple rules: two links in series and two links in parallel. These two rules can be used to estimate the blocking in any complex combination of links.

The next chapter presents fundamentals of the transport layer and end-to-end protocols. The transport layer is responsible for signaling and file transfer.

7.8 Exercises

1. Consider a packet-switching network for server communications with n nodes. For each of the following topologies, sketch the network, and give the average number of hops between a pair of servers (include the server-to-node link as a hop):
   (a) *Star:* one central node with all servers attached to the central node
   (b) *Bidirectional ring:* each node connects to two other nodes to form a loop, with each node connected to one server
   (c) *Fully connected:* each node is directly connected to all other nodes, with each node connected to one server
   (d) *A ring with* $n - 1$ *nodes connected to a central node,* with each node in the ring connected to one server.

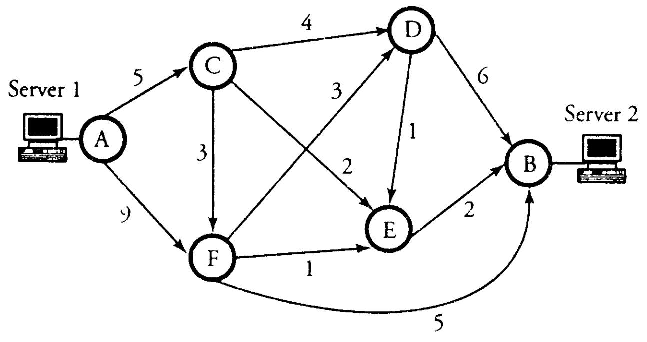

Figure 7.18 Exercises 2–4 network example

2. Figure 7.18 shows a network.
   (a) Find the least-cost path between the two servers, using Dijkstra's algorithm.
   (b) Show an iteration graph as the least-cost path is developed.
3. For the network shown in Figure 7.18:
   (a) Find the least-cost path between the two servers, using the Bellman-Ford algorithm.
   (b) Show an iteration graph as the least-cost path is developed.
4. Consider again the network in Figure 7.18. Assume that each link is bidirectional and that the cost of either direction is identical.
   (a) Find the least-cost path between the two servers, using Dijkstra's algorithm.
   (b) Show an iteration graph as the least-cost path is developed.
5. The network shown in Figure 7.19 is a snapshot of a practical consisting of seven routers. The load on each link is normalized to a number indicated on that link.
   (a) Find the least-cost path between the two routers R1 and R7, using Dijkstra's Algorithm.
   (b) Show an iteration graph as the least-cost path is developed.
6. For the network shown in Figure 7.19:
   (a) Find the least-cost path between the two servers, using the Bellman-Ford algorithm.
   (b) Show an iteration graph as the least-cost path is developed.
7. The practical network shown in Figure 7.20 is a WAN consisting of seven routers R_1 through R_7 interconnecting four LANs. The load on each link is normalized

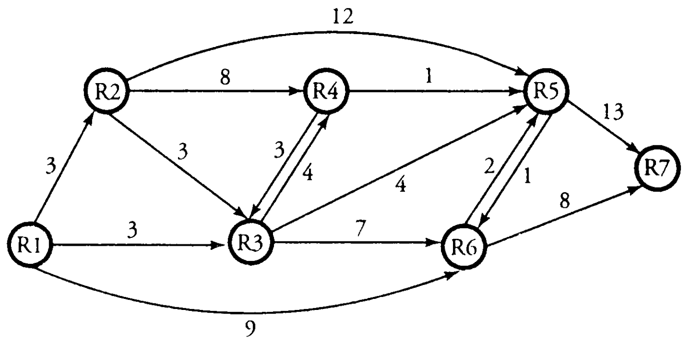

Figure 7.19 Exercises 5–6 network example

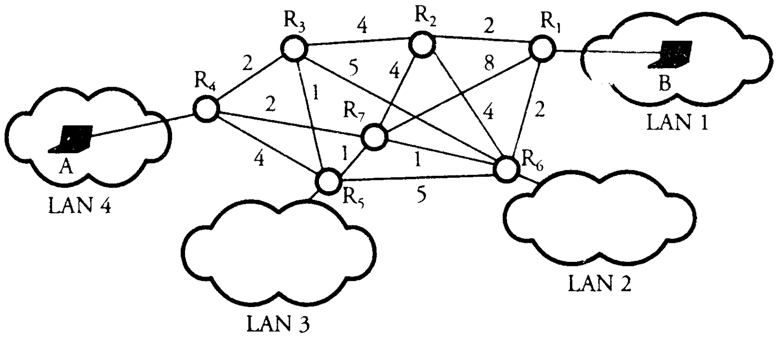

Figure 7.20 Exercise 7 network example

to a number indicated on that link. Assume that all links are bidirectional with equal indicated load.

(a) Find the least-cost path between the two routers R1 and R_7 connecting users A and B, using Dijkstra's algorithm.

(b) Show an iteration graph as the least-cost path is developed.

8. The practical network shown in Figure 7.20 is a WAN consisting of seven routers R_1 through R_7 interconnecting four LANs. The load on each link is normalized to a number indicated on that link. Assume that all links are bidirectional with equal indicated load.

(a) Find the least-cost path between the two routers R1 and R_7 connecting users A and B, using Bellman-Ford's algorithm.

(b) Show an iteration graph as the least-cost path is developed.

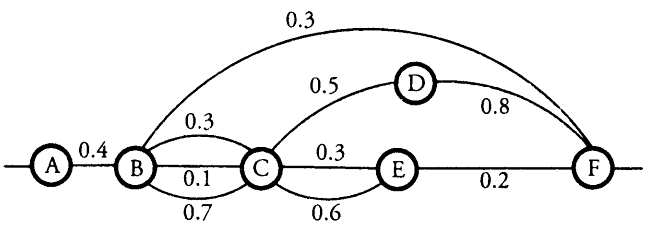

Figure 7.21 Exercise 8 network of links

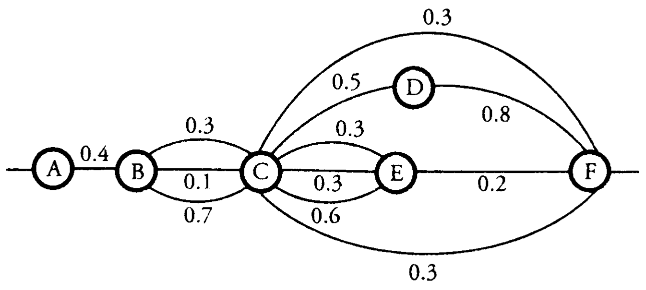

Figure 7.22 Exercise 9 network of links

9. The network in Figure 7.21 has six interconnected nodes, A through F. The corresponding blocking probabilities of links are indicated on the corresponding links. For this network, find the overall blocking probability from A to F.

10. Figure 7.22 shows a large communication network of six interconnected nodes, A through F. The corresponding blocking probabilities of links are indicated on the corresponding links. For this network, find the overall blocking probability from A to F.

11. *Computer simulation project.* Write a computer program to simulate Dijkstra's algorithm for the seven-node network shown in Figure 7.20. Your program must
    (a) Use a scheme that runs for any hypothetical link cost. Users A and B use the Bellman-Ford algorithm.
    (b) Compute the least-cost path between any two routers.

12. *Computer simulation project.* Write a computer program to simulate the Bellman-Ford algorithm for the seven-node network shown in Figure 7.20. Your program must
    (a) Use a scheme that runs for any hypothetical link cost. Users A and B use the Bellman-Ford Algorithm.
    (b) Compute the least-cost path between any two routers.

CHAPTER 8

Transport and End-to-End Protocols

So far, we have covered networking activities at the physical, link, and network layers of the Internet Protocol stack. This chapter focuses on foundations of layer 4, the transport layer. We study several techniques for Transmission Control Protocol (TCP) congestion control. These techniques use a form of end-to-end congestion control in a TCP session when a sender sends a packet and a receiver acknowledges receipt of the packet. This chapter covers the following topics:

- *Transport layer*
- *Transmission Control Protocol (TCP)*
- *User Datagram Protocol (UDP)*
- *Mobile Transport Protocols*
- *TCP Congestion Control*

We first take a close look at *layer 4, the transport layer*, and explain how a file is transferred. Layer 4 handles the details of data transmission and acts as an interface protocol between a communicating host and a server through a network. We explain reliable end-to-end connection establishment under the *Transmission Control Protocol* (TCP).

Next, we present the *user datagram protocol* (UDP), a connectionless transport-layer protocol that is placed on top of the network layer. We also discuss TCP congestion control. Normally, it would take a significant amount of engineering effort to design a network with low or no congestion. *Congestion* in communication networks represents a state in which the traffic flow exceeds the availability of network resources to service the flow. Generally, congestion occurs at a network because of a lack of resources at links and devices. We distinguish between these two categories of resource deficits and explore precautionary solutions: *flow control* and *congestion control*, respectively. We end the chapter with a discussion of TCP and UDP applications.

8.1 Transport Layer

The *transport layer* handles the details of data transmission. This layer provides a logical communication between application processes, as shown in Figure 8.1. Transport-layer protocols are implemented in the end points but not in network routers. The transport layer ensures a complete data transfer for an application process, free from the issue of physical infrastructure. This layer clearly acts as an interface protocol between a communicating host and a network. For example, a client host in LAN 2 is running an application on a server through a *logical* link. However, the end points can be on the opposite sides of the network, connected via a number of routers, as shown by the two end points via R4-R7-R1.

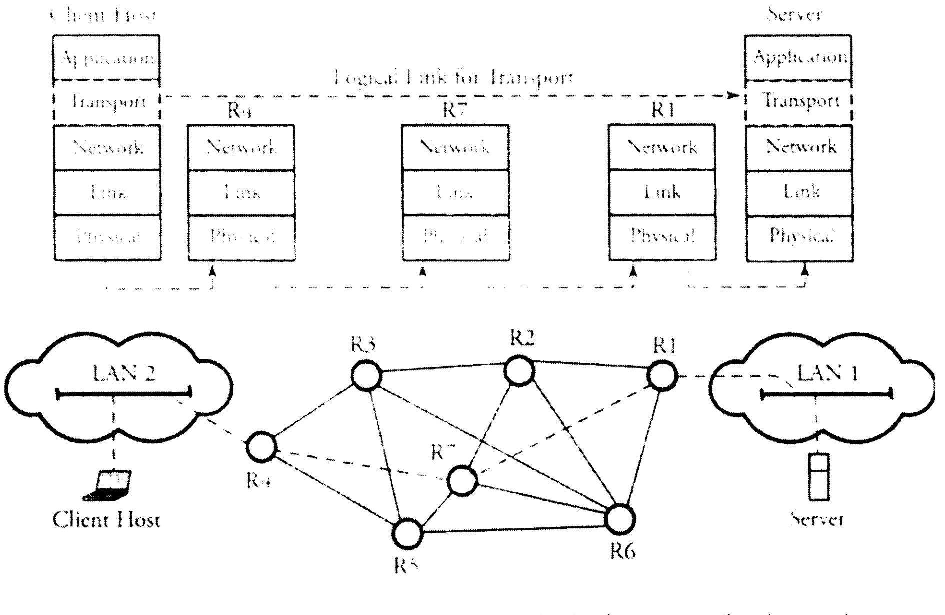

Figure 8.1 Demonstrating "logical" end-to-end communication between a client host and a server at the transport layer

The two most important forms of transportation in the TCP/IP network are governed by two protocols:

- Transmission Control Protocol (TCP)
- User Datagram Protocol (UDP)

TCP is connection oriented and is therefore a reliable service to an invoking application. On the contrary, UDP is a *connectionless* service and is unreliable to the invoking application. A network application design involves making a careful choice between these two transport protocols, as each acts differently to any invoking application.

The transport layer converts application messages into transport-layer packets known as *segments*. The layer manages all end-to-end communication details, such as packet segmentation, packet reassembly, and error checking. The segmentation is performed by breaking the application messages into smaller chunks and attaching a transport-layer header to each chunk to create a segment. Once a segment is formed, it is passed to the network layer, where it is encapsulated within a network-layer packet—a datagram—and transmitted to the destination.

8.1.1 Interaction of Transport and Network Layers

The direct logical link shown in Figure 8.1 reflects the fact that three routers—R4, R7, and R1 act only on the network-layer fields of the datagram and never inspect the fields of the transport-layer segment. This resembles a direct logical, but not physical, link between the two end points. Once the segment is received by the end server, the network layer extracts the transport-layer segment from the datagram and makes it available to the transport layer. At the receiving end, the transport layer processes the segment by extracting the application data in the segment.

Transport protocols can generally offer reliable data transfer service to a certain application in spite of the fact that the underlying network protocol can be unreliable owing to losing packets. A transport protocol can also use its own security procedures to guarantee that application messages are not intruded by unknowns.

8.2 Transmission Control Protocol (TCP)

The *Transmission Control Protocol* (TCP) is a transport-layer protocol that provides a reliable service by using an *automatic repeat request* (ARQ). In addition, TCP provides congestion control, using a sliding window scheme, that introduced in section 4.6.2.

TCP is built over IP services by facilitating a two-way connection between the host application and the destination application.

As a connection-oriented protocol, TCP requires a connection to be established between each two applications. A connection is set up by defining variables the protocol requires and storing them in the *transmission control block*. After establishing the connection, TCP delivers packets in sequence and in bytestream. TCP can either send the data received from the application layer as a single packet or split it into multiple packets and transmit them if an underlying physical network poses a limitation.

The requisites of layer 4 are to transfer data without errors to make sure that packets follow the same sequence. A host application that needs to send a certain sort of data stores it in a send buffer. The data is then transmitted as a bytestream. The transmitting host creates a *segment* that contains a sequence of bytes. The segment is appended to a TCP header that specifies the destination application port and a *sequence number*. When it arrives at the destination, the segment is verified for its integrity. After making sure that the packet is not a duplicate and that the segment number lies in the range of the local buffer, the receiver accepts the packet. The receiver can accept packets out of order. In that case, the receiver simply repositions them in the buffer to obtain an in-order sequence before making the data available to the application. Acknowledgments are cumulative and traverse in the reverse direction.

8.2.1 TCP Segment

As defined earlier, a TCP *segment* is a TCP session packet containing part of a TCP bytestream in transit. The fields of the TCP header segment are shown in Figure 8.2. The TCP segment contains a minimum of 20 bytes of fixed fields and a variable-length options field. The details of the fields are as follows.

- *Source port* and *destination port* specify, respectively, the user's port number, which sends packets and the user's port number, which receives packets.
- *Sequence number* is a 32-bit filed that TCP assigns to each first data byte in the segment. The sequence number restarts from 0 after the number reaches $2^{32} - 1$.
- *Acknowledgment number* specifies the sequence number of the next byte that a receiver waits for and acknowledges receipt of bytes up to this sequence number. If the SYN field is set, the acknowledgment number refers to the *initial sequence number* (ISN).
- *Header length* (HL) is a 4-bit field indicating the length of the header in 32-bit words.

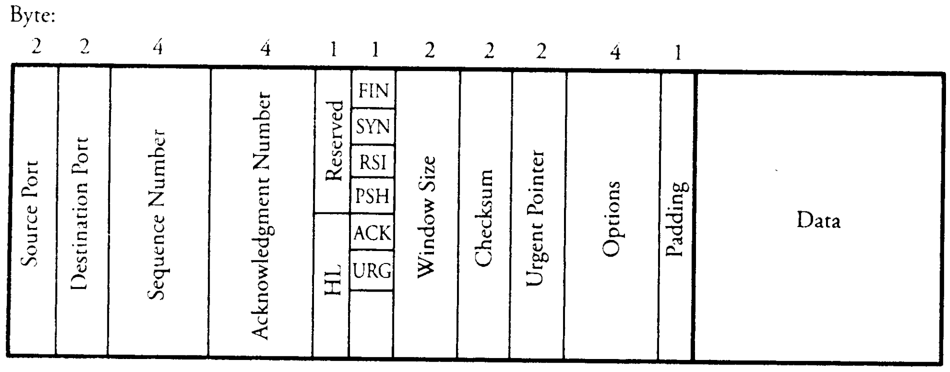

Figure 8.2 TCP segment format

- *Urgent* (URG) is a 1-bit field implying that the urgent-pointer field is applicable.
- *Acknowledgment* (ACK) shows the validity of an acknowledgment.
- *Push* (PSH), if set, directs the receiver to immediately forward the data to the destination application.
- *Reset* (RST), if set, directs the receiver to abort the connection.
- *Synchronize* (SYN) is a 1-bit field used as a connection request to synchronize the sequence numbers.
- *Finished* (FIN) is a 1-bit field indicating that the sender has finished sending the data.
- *Window size* specifies the advertised window size.
- *Checksum* is used to check the validity of the received packet. (See Section 4.5.2 for details.)
- *Urgent pointer* (URG), if set, directs the receiver to add up the values in the urgent-pointer field and the sequence number field to specify the last byte number of the data to be delivered urgently to the destination application.
- *Options* is a variable-length field that specifies the functions that are not available as part of the basic header.

One of the possible *options* is *maximum segment size* (MSS). A receiver uses this option to specify the maximum segment size it can receive. A total of 16 bits are provided to specify this option. Thus, the maximum segment size is limited to 65,535 bytes minus 20 bytes of TCP header and minus 20 bytes of IP header, resulting in

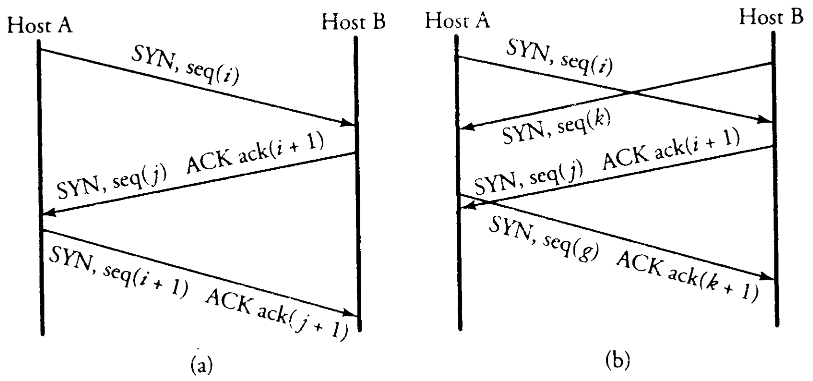
Host A
Host B
SYN, seq(i)
SYN, seq(j) ACK ack(i + 1)
SYN, seq(i + 1) ACK ack(j + 1)
(a)
Host A
Host B
SYN, seq(i)
SYN, seq(k)
SYN, seq(j) ACK ack(i + 1)
SYN, seq(g) ACK ack(k + 1)
(b)

Figure 8.3 TCP signaling: (a) establishment of a connection; (b) collision of two connection requests

65,495 bytes. The default MSS for TCP is 536 bytes. Another possible *option* is the *window scale*. This option is used to increase the size of the advertised window beyond the specified $2^{16} - 1$ in the header. The advertised window can be scaled to a maximum of 2^{14}.

8.2.2 Connection Setup

As a connection-oriented protocol, TCP requires an explicit connection set-up phase. Connection is set up using a three-step mechanism, as shown in Figure 8.3 (a). Assume that host A is a sender and host B a destination. First, the sender sends a connection request to the destination. The connection request comprises the initial sequence number indicated by seq(i), with the SYN bit set. Next, on receipt of the connection request, the destination sends an acknowledgment, ack($i + 1$), back to the source, indicating that the destination is waiting for the next byte. The destination also sends a request packet comprising the sequence number, seq(j), and the SYN bit set. Finally, at the third step, the sender returns an acknowledgment segment, ack($j + 1$), specifying that it is waiting for the next byte. The sequence number of this next segment is seq($i + 1$). This process establishes a connection between the sender and the receiver.

The connection process uses different initial sequence numbers between the sender and the receiver to distinguish between the old and new segments and to avoid the duplication and subsequent deletion of one of the packets. In Figure 8.3, both end-point hosts simultaneously try to establish a connection. In this case, since they recognize the connection requests coming from each other, only one connection is established. If one of the segments from a previous connection arrives late, the receiver accepts the packet, presuming that it belongs to the new connection. If the packet from the current connection with the same sequence number arrives, it is considered a duplicate and is dropped. Hence, it is important to make sure that the initial sequence numbers are different.

When one of the end points decides to abort the connection, a segment with the RST bit set is sent. Once an application has no data to transmit, the sender sends a segment with the FIN bit set. The receiver acknowledges receipt of this segment by responding with an ACK and notifies the application that the connection is terminated. Now, the flow from the sender to the receiver is terminated. However, in such cases, the flow from the receiver to the sender is still open. The receiver then sends a segment with the FIN bit set. Once the sender acknowledges this by responding with an ACK, the connection is terminated at both ends.

8.3 User Datagram Protocol (UDP)

The *user datagram protocol* (UDP) is another transport-layer protocol that is placed on top of the network layer. UDP is a connectionless protocol, as no handshaking between sending and receiving points occurs before sending a segment. UDP does not provide a reliable service. The enhancement provided by UDP over IP is its ability to check the integrity of flowing packets. IP is capable of delivering a packet to its destination but stops delivering them to an application. UDP fills this gap by providing a mechanism to differentiate among multiple applications and deliver a packet to the desired application. UDP can perform error detection to a certain extent but not to the level that TCP can.

8.3.1 UDP Segment

The format of the UDP segment is shown in Figure 8.4. The segment starts with the *source port*, followed by the *destination port*. These port numbers are used to identify the ports of applications at the source or the destination, respectively. The source port identifies the application that is sending the data. The destination port helps UDP to demultiplex the packet and directs it to the right application. The *UDP length* field indicates the length of the UDP segment, including both the header and the data. *UDP checksum* specifies the computed checksum when transmitting the packet from the host. If no checksum is computed, this field contains all zeroes. When this segment is received at the destination, the checksum is computed; if there is an error, the packet is discarded.

Byte:				
2	2	2	2	
Source Port	Destination Port	UDP Length	UDP Checksum	Data

Figure 8.4 User datagram protocol segment

UDP takes messages from the application process, attaches source and destination port number fields and two other fields, and makes this segment available to the network layer. The network layer encapsulates the segment into an IP datagram (packet) and finds the best path to deliver the segment to the other end host.

8.3.2 Applications of TCP and UDP

Although TCP provides a reliable service and UDP does not, many applications fit better in the communication system by using UDP, for the following reasons:

- *Faster delivery of the application object.* TCP is equipped with congestion-control mechanism and requires the guarantee of data delivery at any timing cost. This may not be well suited for real-time applications.
- *Support for a larger number of active clients.* TCP maintains connection state and does track connection parameters. Therefore, a server designated to a particular application supports fewer active clients when the application runs over TCP rather than over UDP.
- *Smaller segment header overhead.* UDP has only 8 bytes of overhead, whereas the TCP segment has 20 bytes of header.

The first category includes such applications as *e-mail, the Web, remote terminal access,* and *file transfer*. These applications run over TCP, as they all require reliable data-transfer service. The second category of applications, including *DNS*, *RIP routing table updates*, and SNMP network management, run over UDP rather than over TCP. For example, RIP updates are transmitted periodically, and reliability may not be an issue, as a more recent update can always replace a lost one.

8.4 Mobile Transport Protocols

In wireless mobile networks, both UDP and TCP have their own applications. However, some modifications are needed in these protocols to become appropriate for wireless networks. One obvious reason for the modification requirement is the fact that a user (host) may move to a remote area and a seamless connection still is desirable.

8.4.1 TCP for Mobility

TCP provides a reliable data delivery owing to the feature of its connection-oriented nature. The most challenging aspect of providing TCP services to a mobile host is the prevention of disruption caused by poor wireless link quality. A poor link quality typically causes to lose TCP data segments leading to a possible timeout. If the poor quality wireless channel is persistent for even a short persistent, the window remains small causing a low throughput.

One option to solve this problem is to disallow a sender to shrink its congestion window when packets are lost for any reason. If a wireless channel soon recovers from disconnection, the mobile host begins to receive data immediately. There are a few other protocols for this purpose among which the *Indirect Transmission Control Protocol* (I-TCP), and the *fast transmit* will be the focus of our discussion.

Indirect TCP

Assume two hosts, one mobile and the other one fixed, are trying to establish an I-TCP connection as shown in Figure 8.5. The I-TCP scheme first splits the connection into two separate connections. One connection is established between the mobile host and the mobile switching center (MSC) which normally attached to the wireless base station, and the other between the MSC and the fixed host. So a connection consists of two pieces: the wireless link and the fixed link. At this point, the wireless and wired link-characteristics remain hidden from the transport layer. The separation of connections into two distinct parts is advantageous to the MSC to better manage communication overheads for a mobile host. This way the throughput of connection is enhanced since the mobile host may be near the from the MSC.

A TCP connection on the wireless portion can support disconnections, and user mobility in addition to wired TCP features such as notification to higher layers on changes in the available bandwidth. Also, the flow control and congestion control mechanisms on the wireless link remain separated from those on the wired link. In the I-TCP scheme, the TCP acknowledgments are separate for the wireless and the wired links of the connection. This resembles two different TCP connections linked

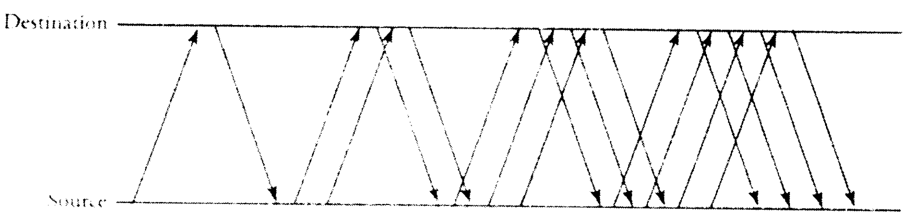

Figure 8.6 Additive increase control for TCP congestion control

unacknowledged data that a sender is allowed to send. Let w_a be the advertised window sent by the receiver, based on its buffer size. thus,

$$w_m = \min(w_g, w_a). \tag{8.3}$$

By having w_m replace w_a, a TCP source is not permitted to transmit faster than the network or the destination. The challenge in TCP congestion control is for the source node to find a right value for the congestion window. The congestion window size varies, based on the traffic conditions in the network. TCP watches for timeout as a sign of congestion. One can arrange timeouts to be used as acknowledgments to find the best size for the congestion window. This is done because the implications for having too large a window are much worse than having too small a window. This TCP technique requires that the timeout values be set properly. Two important factors in setting timeouts follow.

1. Average *round-trip times* (RTTs) and RTT standard deviations are based to set timeouts.
2. RTTs are sampled once every RTT is completed.

Figure 8.6 depicts the additive-increase method. The congestion window is interpreted in terms of packets rather than bytes. Initially, the source congestion window is set to one packet. Once it receives an acknowledgment for the packet, the source increments its congestion window by one packet. So the source transmits two packets at that point. On successful receipt of both acknowledgments, the source once again increments the congestion window by one packet (additive increase).

In practice, the source increments its congestion window by a small amount for each acknowledgment instead of waiting for both acknowledgments to arrive. If a timeout occurs, the source assumes that congestion is developing and therefore sets the congestion window size to half its previous value (multiplicative decrease). The minimum congestion window size is called maximum *segment* size, which represents

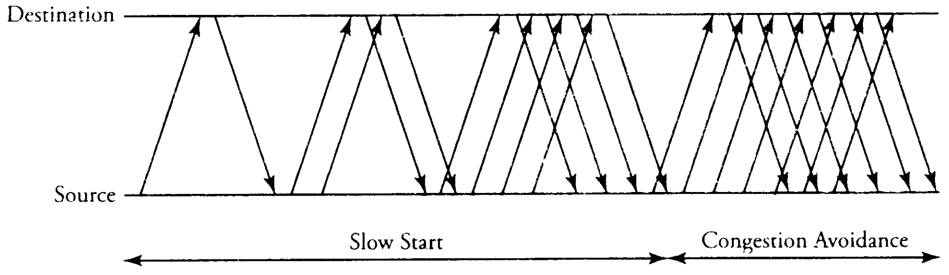

Figure 8.7 Slow-start timing between a source and a destination

one packet. In general, a TCP segment is defined as a TCP session packet containing part of a TCP bytestream in transit.

8.5.2 Slow Start Method

Additive increase is ideally suited when the network operates near capacity. Initially, it would take a considerable amount of time to increase the congestion window. The *slow-start method* increases the congestion window size nonlinearly and in most cases exponentially, as compared to the linear increase in additive increase. Figure 8.7 shows the slow-start mechanism. In this case, the congestion window is again interpreted in packets instead of bytes.

A source initially sets the congestion window to one packet. When its corresponding acknowledgment arrives, the source sets the congestion window to two packets. Now, the source sends two packets. On receiving the two corresponding acknowledgments, TCP sets the congestion window size to 4. Thus, the number of packets in transit doubles for each round-trip time. This nonlinearity trend of increase in the window size continues as seen in the figure. With this method of congestion control, routers on a path may not be able to service the flow of traffic, as the volume of packets increases nonlinearly. This congestion-control scheme by itself may lead to a new type of congestion. The slow-start method is normally used

1. just after a TCP connection is set up or
2. when a source is blocked, waiting for a timeout.

A new variable, *congestion threshold*, is defined. This variable is a saved size of the congestion window when a timeout arrives. When a timeout occurs, the threshold is set to half the congestion window size. Then the congestion window is reset to one packet and ramped up all the way to the congestion threshold, using the slow-start method. Once the connection is established, a burst of packets is sent during a slow

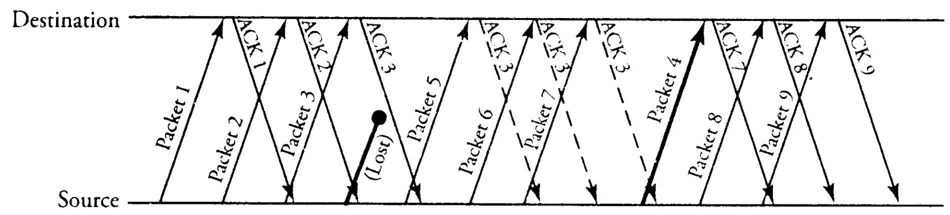

Figure 8.8 Timing of retransmit method between a source and a destination

start. Then, a number of packets are lost, and the source admits no waiting time for acknowledgments. Finally, a timeout occurs, and the congestion window is reduced. Thus, a timeout results in the reduction of the congestion window, as in the previous scheme. The congestion threshold and the congestion window are reset. Slow start is used to increase the congestion window size exponentially.

After the congestion threshold is reached, additive increase is used. At this point, packets may be lost for a while, and the source removes the waiting time for acknowledgments. Then, a timeout occurs immediately, and the congestion window size is reduced. The congestion threshold is reset, and the congestion window is reset to one packet. Now, the source uses slow start to ramp up, and then additive increase is used. After reaching the congestion threshold, additive increase is used. This pattern continues, creating a pulse-type plot. The reason for the large packet loss initially with slow start is that it is more aggressive in the beginning in order to learn about the network. This may result in a few packet losses, but it seems to be better than the conservative approach, in which the throughput is very small.

8.5.3 Fast Retransmit Method

Fast retransmit is based on the concept of duplicate acknowledgment (ACK). The additive-increase and slow-start mechanisms have idle periods, during which the source admits no waiting time for an acknowledgment. Fast retransmit of segments sometimes leads to a retransmission of the lost packet before the associated timeout periods.

Each time it receives an out-of-order packet, the destination should respond with a duplicate ACK of the last successful in-order packet that has arrived. This must be done even if the destination has already acknowledged the packet. Figure 8.8 illustrates the process. The first three packets are transmitted, and their acknowledgments are received.

Now, assume that packet 4 is lost. Since it is not aware of this lost packet, the source continues to transmit packet 5 and beyond. However, the destination sends the duplicate acknowledgment of packet 3 to let the source know that it has not received packet 4. In practice, once the source receives three duplicate acknowledgments, it retransmits the lost packet. Fast recovery is another improvement to TCP congestion control. When congestion occurs, instead of dropping the congestion window to one packet, the congestion window size is dropped to half, and additive increase is used. Thus, the slow-start method is used only during the initial connection phase and when a timeout occurs. Otherwise, additive increase is used.

8.5.4 TCP Congestion Avoidance Methods

Network congestion is a traffic bottleneck between a source and a destination. *Congestion avoidance* uses precautionary algorithms to avoid possible congestion in a network. Otherwise, TCP congestion control is applied once congestion occurs in a network. TCP increases the traffic rate to a point where congestion occurs and then gradually reduces the rate. It would be better if congestion could be avoided. This would involve sending some precautionary information to the source just before packets are discarded. The source would then reduce its sending rate, and congestion could be avoided to some extent.

Source-Based Congestion Avoidance

Source-based congestion avoidance detects congestion early from end-hosts. An end host estimates congestion in the network by using the round-trip time and throughput as it measures. An increase in round-trip time can indicate that routers' queues on the selected routing path are increasing and that congestion may happen. The source-based schemes can be classified into four basic algorithms:

1. *Use of round trip time (RTT) as a measure of congestion in the network.* As queues in the routers build up, the RTT for each new packet sent out on the network increases. If the current RTT is greater than the average of the minimum and maximum RTTs measured so far, the congestion window size is reduced.
2. *Use of RTT and window size to set the current window size.* Let w be the current window size, w_o be the old window size, r be the current RTT, and r_o be the old RTT. A window RTT product is computed based on $(w - w_o)(r - r_o)$. If the product is positive, the source decreases the window size by a fraction of its old value. If the product is negative or 0, the source increases the window size by one packet.
3. *Use of throughput as a measure to avoid congestion.* During every RTT, a source increases the window size by one packet. The achieved throughput is then compared

with the throughput when the window size was one packet smaller. If the difference is less than half the throughput at the beginning of the connection when the window size was one packet, the window size is reduced by one packet.

4. *Use of throughput as a measure to avoid congestion.* But this time, the algorithm uses two parameters: the current throughput and the expected throughput to avoid congestion.

TCP normalized is presented next, as an example for the fourth algorithm.

TCP Normalized Method

In the *TCP normalized method*, the congestion window size is increased in the first few seconds, but the throughput remains constant, because the capacity of the network has been reached, resulting in an increase in the queue length at the router. Thus, an increase in the window size results in any increase in the throughput. This traffic over and above available bandwidth of the network is called *extra data.* The idea behind TCP normalized is to maintain this extra data at a nominal level. Too much of extra data may lead to longer delays and congestion. Too little extra data may lead to an underutilization of resources, because the available bandwidth changes owing to the bursty nature of Internet traffic. The algorithm defines the expected value of the rate $E[r]$ as

$$E[r] = \frac{w_g}{r_m}, \tag{8.4}$$

where r_m is the minimum of all the measured round-trip times, and w_g is the congestion window size. We define A_r as the actual rate and $(E[r] - A_r)$ as the rate difference. We also denote the maximum and minimum threshold to be ρ_{max} and ρ_{min}, respectively. When the rate difference is very small—less than ρ_{min}—the method increases the congestion window size to keep the amount of extra data at a nominal level. If the rate difference is between ρ_{min} and ρ_{max}, the congestion window size is unaltered. When the rate difference is greater than ρ_{max}, there is too much extra data, and the congestion window size is reduced. The decrease in the congestion window size is linear. The TCP normalized method attempts to maintain the traffic flow such that the difference between expected and actual rates lies in this range.

8.6 Summary

All end-to-end communications, such as packet segmentation, packet reassembly, and error checking are managed through the *transport layer*. The transport layer is responsible for connection signaling and file transfer. The two most important forms of transport are the *Transmission Control Protocol* (TCP) and the *User Datagram Protocol* (UDP).

TCP provides a reliable service. Reliability is provided using an *automatic repeat request* (ARQ) protocol. TCP also provides flow control using a sliding-window scheme, as well as congestion control. *Additive increase, multiplicative decrease control* performs a slow increase in the congestion window size when network congestion decreases and a fast drop in the window size when congestion increases. A better technique is the *slow-start method*, which increases the congestion window size nonlinearly. The *fast retransmit* of segments sometimes leads to a retransmission of the lost packet before the associated timeout periods. We looked at various congestion-avoidance mechanisms, including the TCP normalized mechanism.

UDP is another transport-layer protocol described in this chapter. UDP is placed on top of the network layer and is a connectionless protocol, as there is no handshaking between sending and receiving points before sending a segment.

The next chapter presents the application layer in the protocol stack. This layer is responsible for networking applications, *e-mail* and the *World Wide Web* (WWW). The chapter ends with a discussion of network management.

8.7 Exercises

1. Consider a wide area network in which two hosts, A and B, are connected through a 100 km communication link with the data speed of 1 Gb/s. Host A wants to transfer the content of a CD-ROM with 200 Kb of music data while host B reserves portions of its 10 parallel buffers, each with the capacity of 10,000 bits. Use Figure 8.3 and assume that host A sends a SYN segment, where ISN = 2,000 and MSS = 2,000 and that host B sends ISN = 4,000 and MSS = 1,000. Sketch the sequence of segment exchange, starting with host A sending data at time $t = 0$. Assume host B sends ACK every five frames.
2. Consider that host 1 transfers a large file of size f to host 2 with MSS = 2,000 bytes over a 100 Mb/s link.
   (a) Knowing that the TCP sequence number field has 4 bytes, find f such that TCP sequence numbers are not exhausted.
   (b) Find the time it takes to transmit f. Include the link, network, and transport headers attached to each segment.
3. Assume that a TCP connection is established. Find the number of round-trip times the connection takes before it can transmit n segments, using
   (a) Slow-start congestion control

(b) Additive increase congestion control

4. We want to understand the fairness index of resource allocations. Suppose that a congestion-control scheme can face five possible flows with the following through-put rates: B_1 = 1 Gb/s, B_2 = 1 Gb/s, B_3 = 1 Gb/s, B_4 = 1.2 Gb/s, and B_5 = 16 Gb/s.
   (a) Calculate the fairness index for this scheme for B_1, B_2, and B_3.
   (b) What useful information does the result of part (a) provide?
   (c) Now, consider all five flows, and calculate the fairness index for this scheme.
   (d) What would the result of part (c) mean to each flow?

5. Assume that a TCP connection is established over a moderately congested link. The connection loses one packet every five segments (packets).
   (a) Can the connection survive at the beginning with the linear portion of congestion avoidance?
   (b) Assume that the sender knows that the congestion remains in the network for a long time. Would it be possible for the sender to have a window size greater than five segments? Why?

6. A TCP connection is established over a 1.2 Gb/s link with a round-trip time of 3.3 ms. To transmit a file of size 2 MB, we start sending it, using 1 KB packets.
   (a) How long does the transmission take if an additive increase, multiplicative decrease control with a window size of w_g = 500 KB is used?
   (b) Repeat part (a), using slow-start control.
   (c) Find the throughput of this file transfer.
   (d) Find the bandwidth utilization for this transfer.

7. Consider that an established TCP connection has a round-trip time of approximately 0.5 second and forms a window size of w_g = 6 K. The sending source transmits segments (packets) every 50 ms, and the destination acknowledges each segment every 50 ms. Now assume that a congestion state develops in this connection such that the destination does not receive a segment. This loss of segment is detected by the fast-retransmit method at the fourth receipt of duplicate ACK.
   (a) Find the amount of time the sending source has lost if the source uses the arrival of duplicate ACKs as a sign for moving the window forward one segment.
   (b) Repeat part (a), this time under a condition that the sending source waits to receive the ACK of the retransmitted packet before moving the window forward one segment.

CHAPTER 9
Applications and Network Management

Having presented basic concepts and definitions in networking to the point that an end-to-end connection can be established, we now look at the last layer of the protocol stack—the *application layer*—and certain network-management issues. This chapter examines Internet applications and their supporting protocols and services. These services demonstrate how users perceive a computer network and express the power of Internet technology. This chapter focuses on the following topics:

- *Application-layer overview*
- *Domain Name System (DNS)*
- *Remote login protocols*
- *Electronic mail (e-mail)*
- *File transfer and FTP*
- *World Wide Web (WWW) and HTTP*
- *Network management*

Layer 5, the *application layer*, determines how a specific user application should use a network. The *Domain Name System* (DNS) server is an essential entity for translating machine or domain names into numerical IP addresses. *Remote login protocols* allow applications to run at a remote site, with results transferred back to their local sites. Two remote login protocols are TELNET and SSH.

The most-applied Internet tool is *e-mail*. The *Simple Mail Transfer Protocol* (SMTP) is one of the protocols for sending *electronic mail* (e-mail) from the mail server of a source to the mail servers of destinations. The *World Wide Web* (WWW), or simply *Web*, is a global network of servers linked together by protocols allowing access to all connected hypertext resources.

Finally, this chapter discusses network management. ASN.1 is a formal language for each managed device to be defined in a network under management. MIB is another management tool for accommodating a database of information and characteristics for devices. The *Simple Network Management Protocol* (SNMP) enables a network manager to find the location of a fault. SNMP runs on top of UDP and uses client/server configurations. SNMP commands define how to query information from a server and forward information to a server or a client.

9.1 Application-Layer Overview

The *application layer* is built on the transport layer and provides network services to user applications. The application layer defines and performs such applications as electronic mail (e-mail), remote access to computers, file transfers, newsgroups, and the Web, as well as streaming video, Internet radio and telephony, P2P file sharing, multiuser networked games, streaming stored video clips, and real-time video conferencing.

The application layer has its own software dependencies. When a new application is developed, its software must be able to run on multiple machines, so that it does not need to be rewritten for networking devices, such as routers, that function at the network layer. In a *client/server* architecture for example, a *client* end host requests services from a *server* host. A client host can be on sometimes or always. Figure 9.1 shows an example of application-layer communication.

9.1.1 Client and Server Model

A *client/server model* provides specific computational services, such as partial-time usage services to multiple machines. Reliable communication protocols, such as TCP, allow interactive use of remote servers as well. For example, we can build a server that provides remote image-processing services to clients. Implementing such a communication service requires a server loaded with the application protocol to accept requests and a client to make such requests. To invoke remote image processing, a user first executes a client program establishing a TCP connection to a server. Then, the client begins transmitting pieces of a raw image to the server. The server processes the received objects and sends the results back.

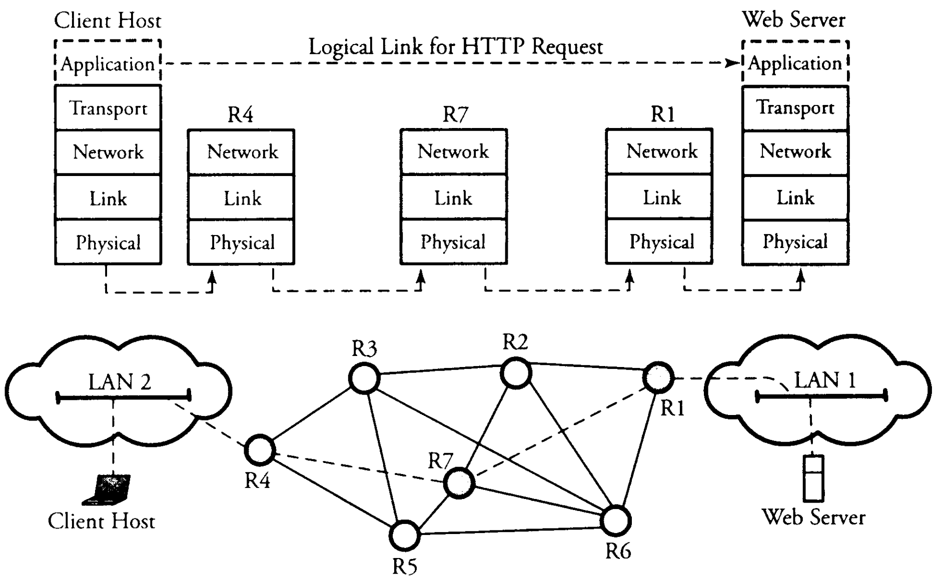

Figure 9.1 Web communication between two end systems

9.2 Domain Name System (DNS)

One of the most important components of the application layer is the *Domain Name System* (DNS) server. DNS is a distributed hierarchical and global directory that translates machine or domain names to numerical IP addresses. DNS can be thought as a distributed database system used to map host names to IP addresses, and vice versa. DNS is a critical infrastructure, and all hosts contact DNS servers when they initiate connections. DNS can run over either UDP or TCP. However, running over UDP is usually preferred, since a fast response for a transaction provided by UDP is required. Some of the information-processing functions a DNS server handles are

- Finding the address of a particular host
- Delegating a subtree of server names to another server
- Denoting the start of the subtree that contains cache and configuration parameters, and giving corresponding addresses
- Naming a host that processes incoming mail for the designated target
- Finding the host type and the operating system information
- Finding an alias for the real name of a host
- Mapping IP addresses to host names

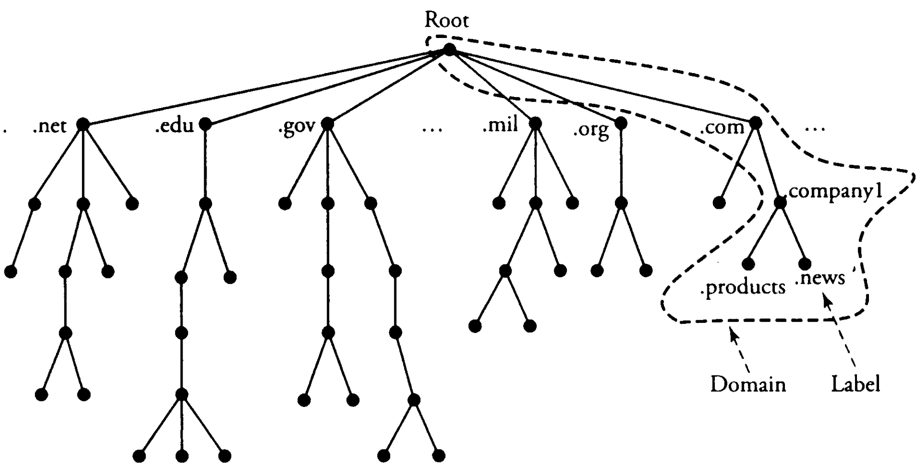

Figure 9.2 Hierarchy of domain name space, labels, and domain names

DNS is an application-layer protocol, and every Internet service provider—whether for an organization, a university campus, or even a residence—has a DNS server. In the normal mode of operation, a host sends UDP queries to a DNS server. The DNS server either replies or directs the queries to other servers. The DNS server also stores information other than host addresses.

The DNS routinely constructs a query message and passes it to the UDP transport layer without any handshaking with the UDP entity running on the destination end system. Then, a UDP header field is attached to the message, and the resulting segment is passed to the network layer. The network layer always encapsulates the UDP segment into a *datagram*. The datagram, or packet, is now sent to a DNS server. If the DNS server does not respond, the fault may be UDP's unreliability.

9.2.1 Domain Name Space

Any entity in the TCP/IP environment is identified by an IP address, which thereby identifies the connection of the corresponding host to the Internet. An IP address can also be assigned a *domain name*. Unique domain names assigned to hosts must be selected from a *name space* and are generally organized in a hierarchical fashion.

Domain names are defined in a tree-based structure with the root at the top, as shown in Figure 9.2. A tree is structured with a maximum of 128 levels, starting at level 0 (root). Each level consists of nodes. A node on a tree is identified by a *label*, with a string of up to 63 characters, except for the root label, which has empty string.

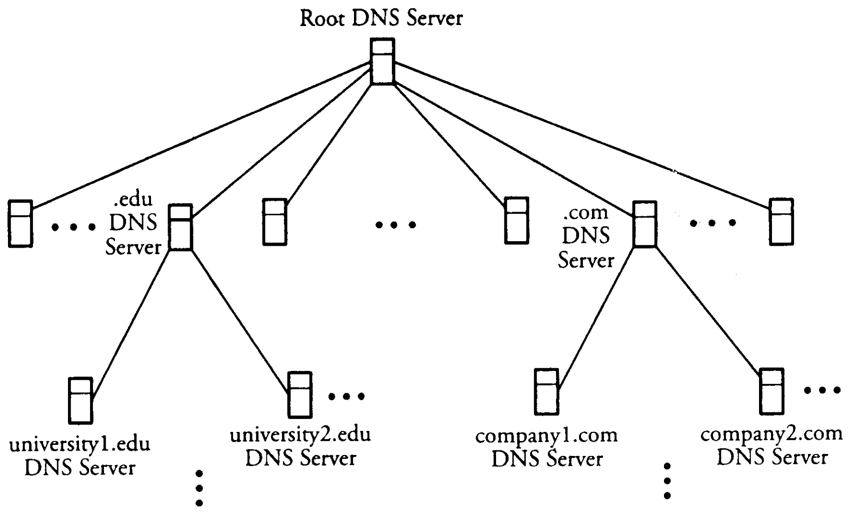

Figure 9.3 Hierarchy of DNS domain name servers

The last label of a domain name expresses the type of organization; other parts of the domain name indicate the hierarchy of the departments within the organization. Thus, an organization can add any suffix or prefix to its name to define its host or resources. A domain name is a sequence of labels separated by dots and is read from the node up to the root. For example, moving from right to left, we can parse as follows: domain name news.company1.com, a commercial organization (.com) and the "news" section of "company1" (news.company1). Domain names can also be partial. For example, company1.com is a partial domain name.

Domain-Name Servers

The domain name space is divided into subdomains, and each domain or subdomain is assigned a ***domain name server***. This way, we can form a hierarchy of servers, as shown in Figure 9.3, just as we did for the hierarchy of domain names. A domain name server has a database consisting of all the information for every node under that domain. Each server at any location in the hierarchy can partition part of its domain and delegate some responsibility to another server. The ***root server*** supervises the entire domain name space. A root server typically does not store any information about domains and keeps references only to servers over which it has authority. Root servers are distributed around the world.

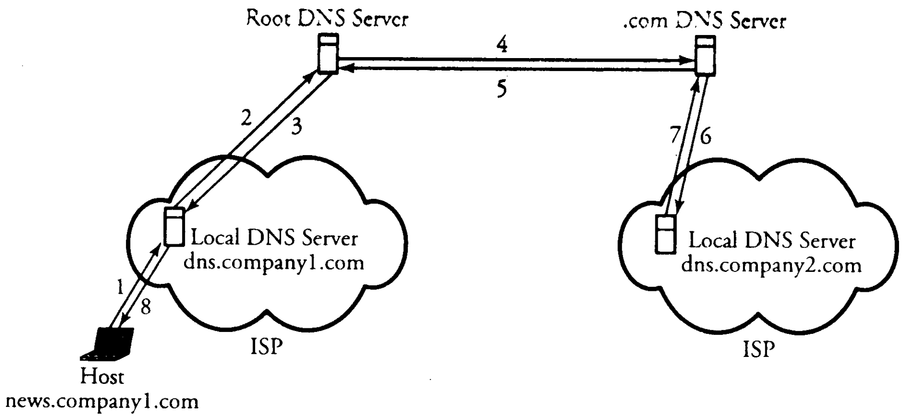

Figure 9.4 Recursive mapping

9.2.2 Name/Address Mapping

DNS operates based on the client/server application. Any client host can send an IP address to a domain name server to be mapped to a domain name. Each host that needs to map an address to a name or vice versa should access the closest DNS server with its request. The server finds and releases the requested information to the host. If the requested information is not found, the server either delegates the request to other servers or asks them to provide the information. After receiving the mapping information, the requesting host examines it for correctness and delivers it to the requesting process.

Mapping can be of either *recursive* or *iterative*. In recursive mapping (Figure 9.4), the client host makes the request to its corresponding DNS server. The DNS server is responsible for finding the answer recursively. The requesting client host asks for the answer through its local DNS server, news.company1.com. Assume that this server contacts the root DNS server, and still the information has not been found. This time, the root DNS server sends the query to the .com server, but the transaction still remains unsuccessful. Finally, .com server sends the query to the local DNS server of the requested place, as dns.company2.com, and finds the answer. The answer to a query in this method is routed back to the origin, as shown in the figure. The local DNS server of the requested place is called the *authoritative server* and adds information to the mapping, called time to live (TTL).

In the iterative approach, the mapping function is as shown in Figure 9.5. In this case, if it does not have the name to provide, the server returns to the client host.

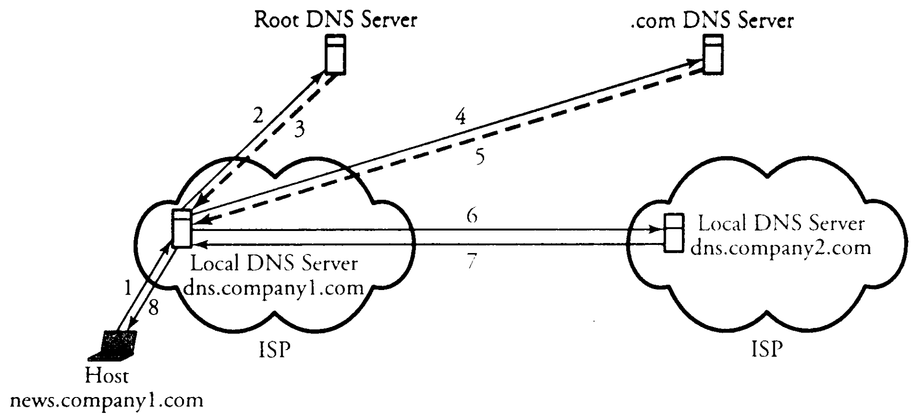

Figure 9.5 Iterative mapping

The host must then repeat the query to the next DNS server that may be able to provide the name. This continues until the host succeeds in obtaining the name. In Figure 9.5, the news.company1.com host sends the query to its own local DNS server, dns.company1.com—thus trying the root DNS server first—and then tries .com server, finally ending up with the local DNS server of the requested place: dns.company2.com.

9.2.3 DNS Message Format

DNS communication is made possible through *query* and *reply* messages. Both message types have the 12-byte header format shown in Figure 9.6. The query message consists of a header and a question message only. The reply message consists of a header and four message fields: *question*, *answer*, *authority*, and *additional information*.

The header has six fields as follows. A client uses the *identification* field to match the reply with the query. This field may appear with a different number each time a client transmits a query. The server copies this number in its reply. The *flags* field contains subfields that represent the type of the message, such as the type of answer requested or requested DNS recursive or iterative mapping. The *number of questions* field indicates how many queries are in the *question* portion of the message. The *number of answers* shows how many answers are in the *answer* field. For the query message, this field contains all zeros. The *number of authoritative records* field consists of the number of authoritative records in the *authority* portion of a reply message. Similarly, this field is filled by zeros for a query message. Finally, the *number of additional records* field records

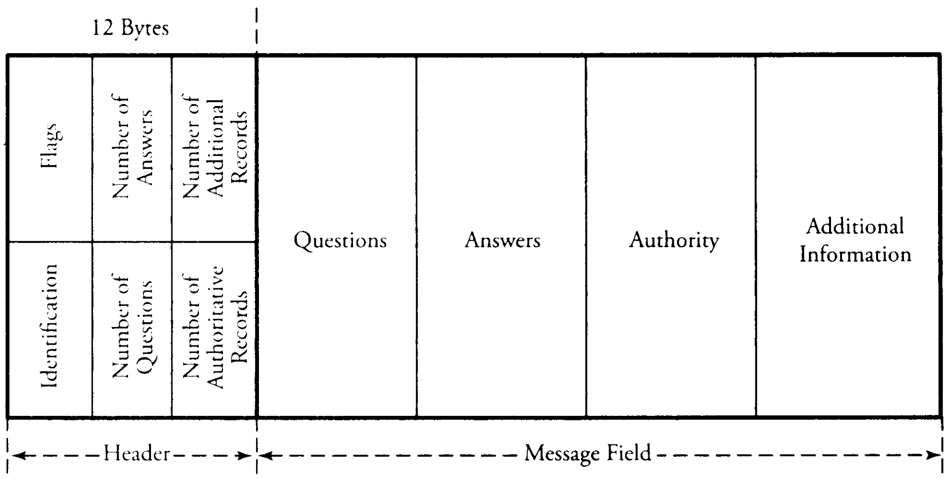

Figure 9.6 DNS message format

are in the additional information portion of a reply message and is similarly filled by zeros in a query message.

The *questions* field can contain one or more questions. The *answers* field belongs only to a reply message and consists of one or more replies from a DNS server to the corresponding client. The *authority* field is present only in reply messages and provides the domain name information about one or more authoritative servers. Finally, the *additional information* field is present only in reply messages and contains other information, such as the IP address of the authoritative server. This additional information helps a client further identify an answer to a question.

The next section explains how domains are added to DNS database. New information and names are included into a DNS database through a *registrar*. On a request for inclusion of a new domain name, a DNS registrar must verify the uniqueness of the name and then enter it into its database.

9.3 Remote Login Protocols

A client/server model can create a mechanism that allows a user to establish a session on the remote machine and then run its applications. This application is known as *remote login*. A user may want to run such applications at a remote site, with results to be transferred back to its local site. For example, an employee working at home can log in to his/her work server to access application programs for doing a project. This can be done by a client/server application program for the desires service. Two remote login protocols are TELNET and SSH.

9.3.1 TELNET Protocol

TELNET (*teletype network*) is a TCP/IP standard for establishing a connection to a remote system. *TELNET* allows a user to log in to a remote machine across the Internet by first making a TCP connection and then pass the detail of the application from the user to the remote machine. This application can be interpreted as if the text being transferred had been typed on a keyboard attached to the remote machine.

Logging to Remote Servers

With TELNET, an application program on the user's machine becomes the client. The user's keyboard and its monitor also attach directly to the remote server. The remote-logging operation is based on timesharing, whereby an authorized user has a login name and a password. TELNET has the following properties.

- Client programs are built to use the standard client/server interfaces without knowing the details of server programs.
- A client and a server can negotiate data format options.
- Once a connection is established through TELNET, both ends of the connection are treated symmetrically.

When a user logs in to a remote server, the client's terminal driver accepts the keystrokes and interprets them as characters by its operating system. Characters are typically transformed to a universal character set called *network virtual terminal* (NVT), which uses 7-bit USASCII representation for data. The client then establishes a TCP connection to the server. Texts in the NVT format are transmitted using a TCP session and are delivered to the operating system of the remote server. The server converts the characters back from NVT to the local client machine's format.

The NVT process is necessary because computers to be remotely logged in to differ. In such cases, a specific terminal emulator must also be used by the TELNET client and servers. The client accepts keystrokes from the user's keyboard while accepting characters that the server sends back. On the server side, data moves up through the server's operating system to the server application program. The remote operating system then delivers characters to the application program the user is running. In the meantime, remote character echo is transmitted back from the remote server to the client over the same path. If the application at the server's site stops reading input for any reason ,the associated operating system and, therefore, the server are overwhelmed.

TELNET also offers several options that allow clients and servers to negotiate on nonroutine transactions. For example, one option is that the client and the server to pass 8-bit data. In such cases, both client and server must agree to pass 8-bit data before any transmission.

9.3.2 Secure Shell (SSH) Protocol

Secure Shell (SSH), another remote login protocol, is based on UNIX programs. SSH uses TCP for communications but is more powerful and flexible than TELNET and allows the user to more easily execute a single command on a remote client. SSH has the following advantages over TELNET.

- SSH provides a secure communication by encrypting and authenticating messages (discussed in chapter 10).
- SSH provides several additional data transfers over the same connection by multiplexing multiple channels that are used for remote login.

SSH security is implemented by using *public-key encryption* between the client and remote server. When a user establishes a connection to a remote server, the data being transmitted remains confidential even if an intruder obtains a copy of the packets sent over an SSH connection. SSH also implements an authentication process on messages so that a server can find out and verify the host attempting to form a connection. Normally, SSH requires users to enter a private password.

A simple SSH interactive session starts with the server's listening on its port specifically designated for secure transmissions. After a password is submitted, SSH starts a shell for the session. SSH can handle several data transfers simultaneously in a same session. This type of remote login service is multiplexed over an SSH connection. SSH can also be used between two machines to carry out *port forwarding* by establishing a secure tunnel. In the SSH remote login utility, a user can allow SSH to automatically splice an incoming TCP connection to a new connection across a tunnel. (The details of tunneling and its applications are explained in Chapter 16.) Data sent over the Internet is ensured to be delivered safe from snooping and alteration.

SSH resembles a tunneling function. For example, when it forms an SSH connection for its port k_1 to a remote server, a client can determine that an incoming TCP connection for this port be automatically forwarded across the tunnel to the server and then spliced to another connection to port k_2 of a second server. This way, the client

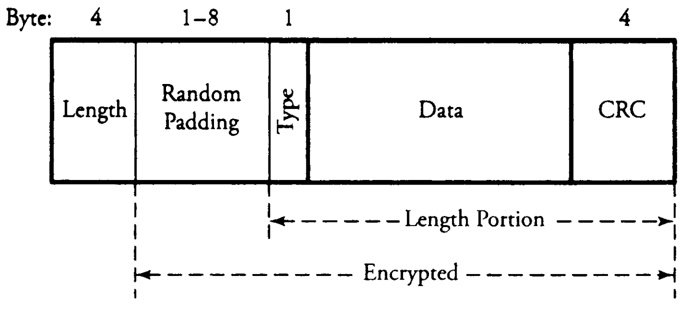

Figure 9.7 SSH packet format

has a TCP connection established on its machine, and the second server makes a TCP connection to the first server. The advantage of port forwarding is that application data can be passed between two sites—the client and the second server—without requiring a second client and server—the first server as a client and the second server.

Figure 9.7 shows the format of an SSH packet.

- *Length* indicates the size of the packet, not including the *length* field or the variable-length *random padding* field that follows it.
- *Padding* causes an intrusion to be more difficult.
- *Type* identifies the type of message.
- *CRC*, or cyclic redundancy check, is an error-detection field (see Chapter 4).

When encryption is enabled, all fields except *length* are encrypted. SSH also permits optional compression of the data, which is useful when SSH is used in low-bandwidth situations. In such cases, the client and the server negotiate compression, and only the *type* and *data* fields are compressed.

9.4 Electronic Mail (E-mail)

9.4.1 Simple Mail Transfer Protocol (SMTP) and E-mail

The *Simple Mail Transfer Protocol* (SMTP) plays a major role in transferring Internet electronic mail. This protocol transfers *electronic mail* (*e-mail*) from the mail server of a source to the mail servers of destinations. SMTP is older than the Hypertext Transfer Protocol (HTTP), the Web communication protocol, and imposes certain restrictions, such as limits on the size of e-mail content.

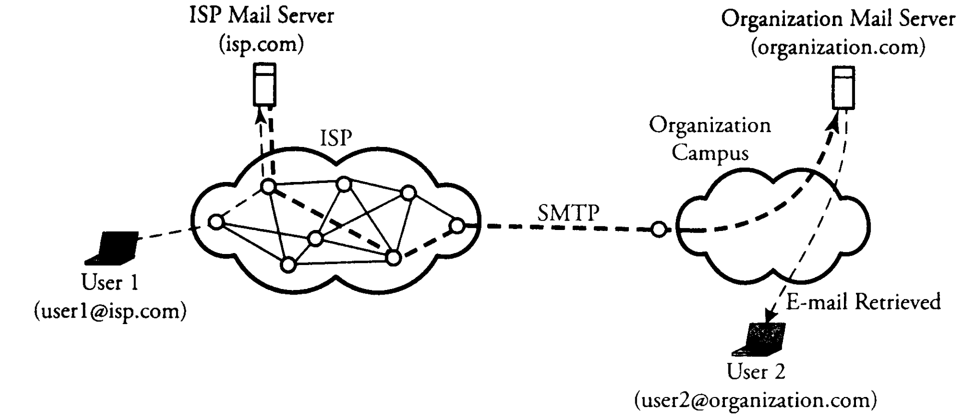

Figure 9.8 Two users exchanging e-mail through SMTP

In Figure 9.8, user 1 is in a residential area, has an Internet service provider (ISP), and is sending an e-mail to user 2, working in an organization. Suppose that the mail servers are isp.com and organization.com, respectively. Thus, user 1 and user 2 have e-mail addresses of user1@isp.com and user2@organization.com, respectively. The procedure for an e-mail exchange between user 1 and user 2 is as follows.

Begin SMTP Between Two Users

1. User 1 provides user 2's e-mail address (user2@organization.com) and composes its message.
2. User 1 sends the message to its mail server (isp.com).
3. Server isp.com places the message in its queue.
4. SMTP on user 1's mail server notices the message in the queue and opens a TCP connection with the organization mail server (organization.com).
5. Initial SMTP handshaking takes place between the two servers.
6. The message is sent to organization.com's mail server, using the established TCP connection.
7. User 2's mail server receives the message and then puts it in user 2's mailbox, ready to be retrieved by user 2. ■

A *user mailbox* is a space in the mail server allocated to the user to keep its e-mail. Also, SMTP is designed to connect only the two mail servers of the associated parties, regardless of the distance between the two users. Consequently, this protocol involves only the two mail servers of the communicating users.

9.5 File Transfer and FTP

File transfer is another computer networking application. It is always essential that files and information geographically distributed over different locations be shared among the members of a working group. In a certain application, files are typically saved in a server. A user then uses a file transfer protocol to access the server and transfer the desired file. Two file transfer protocols are FTP and SCP.

9.5.1 File Transfer Protocol (FTP)

File Transfer Protocol (FTP) is part of the TCP/IP suite and is very similar to TELNET. Both FTP and TELNET are built on the client/server paradigm, and both allow a user to establish a remote connection. However, TELNET provides a broader access to a user, whereas FTP allows access only to certain files. The essence of this protocol is as follows.

Begin File Transfer Protocol

1. A user requests a connection to a remote server.
2. The user waits for an acknowledgment.
3. Once connected, the user must enter a user ID, followed by a password.
4. The connection is established over a TCP session.
5. The desired file is transferred.
6. The user closes the FTP connection. ■

FTP can also run through a Web browser.

9.5.2 Secure Copy Protocol (SCP)

The *Secure Copy Protocol* (SCP) is similar to TELNET but is secure. Incorporated in the SCP structure are a number of encryption and authentication features that are similar to those in SSH. Also similar is the exchange of commands between local and remote hosts. SCP commands automatically prompt the user for the password information when it is time to access a remote machine. SCP cannot handle file transfer between machines of significantly different architectures.

9.6 World Wide Web (WWW) and HTTP

Application-layer software is the intelligence built for end servers. The *World Wide Web* (WWW), or simply *Web*, is a global network of servers linked by a common protocol allowing access to all connected hypertext resources. When a client host requests an object, a Web server responds by sending the requested object through browsing tools. A *browser* is a user agent displaying the requested Web page. The *Hyper Text Transfer Protocol* (HTTP) transfers that page at the application layer. HTTP uses TCP rather than UDP, since reliability of delivery is important for Web pages with text. The TCP connection-establishment delay in HTTP is one of the main contributing delay factors associated with downloading Web documents.

HTTP is based on the client/server idea, having a client and a server program, both of which can be executed on different end systems. The communication is carried out through an exchange of HTTP messages. This protocol specifies the structure of these messages. For example, HTTP defines how a pair of client/server hosts should exchange messages. In this context, a Web page consists of files, such as Hypertext Markup Language (HTML) file or an image that can be addressed by a single *uniform resource locator* (URL). A URL is a global address of an HTML document and has two parts. The first part indicates what protocol is used, and the second part determines the IP address of the associated resource.

9.6.1 Web Caching (Proxy Server)

An HTTP request from a user is first directed to the network *proxy server*, or *Web cache*. Once configured by the network, a browser's request for an object is directed to the Web cache, which must contain updated copies of all objects in its defined proximity. The main reason for Web caching is to reduce the response time for a user request. This benefit is much more obvious when the bandwidth to a requested server is limited because of traffic at certain hours of the day. Normally, each organization or ISP should have its own cache providing a high-speed link to its users. Consequently, it is to users' advantages that this rapid method of finding objects be available. This method of Internet access also reduces traffic on an organization's access link to the Internet. The details of Web caching are as follows:

Begin Web Caching Algorithm

1. The source browser makes a TCP connection to the Web cache.
2. The user browser transmits its HTTP request to the Web cache.
3. If it has a copy of the requested object, the Web cache forwards the object to the user browser.

   Otherwise the Web cache establishes a TCP connection to the requested server and asks for the object. Once it receives the requested object, the Web cache stores a copy of it and forwards another copy to the requesting user browser over the existing TCP connection. ■

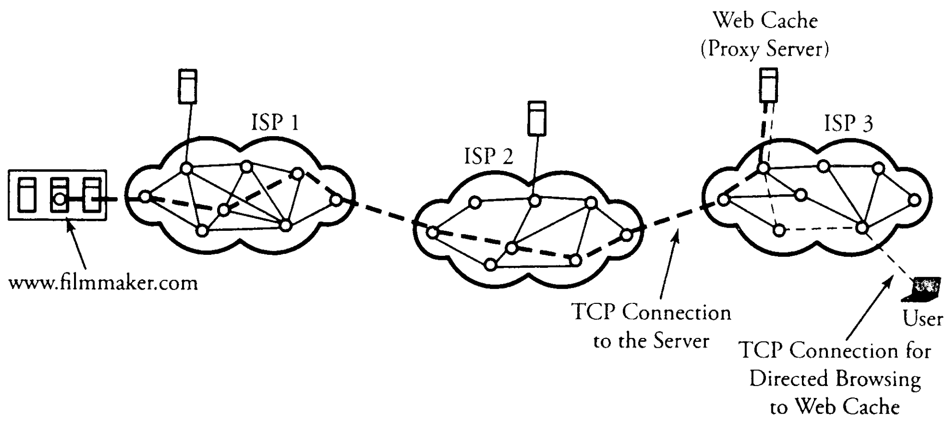

Figure 9.9 A user's browser requesting an object through the Web cache

Figure 9.9 shows three Internet service providers (ISPs). A user in ISP domain 3 is browsing to find and watch an object named http://www.filmmaker.com in ISP domain 1. The request for this object is directed to the Web cache, shown by dashed lines. In this example, the Web cache has no record of the requested object and therefore is establishing another TCP connection to update its record.

9.7 Network Management

The main purpose of *network management* is to monitor, manage, and control a network. A network can be structured with many links, routers, servers, and other physical-layer devices, which can be equipped with many network protocols that coordinate them. Imagine when thousands of such devices or protocols are tied together by an ISP and how drastic their management can become to avoid any interruptions in routine services. In this context the purpose of network management is to monitor, test, and analyze the hardware, software, and human elements of a network and then to configure and control those elements to meet the operational performance requirements of the network.

Figure 9.10 illustrates a simple network management scenario in which LANs connect to the Internet. LAN 1 is dedicated to the network administrator facilities. The network administrator can periodically send management packets to communicate with a certain network entity. A malfunctioning component in a network can also initiate communication of its problem to the network administrator.

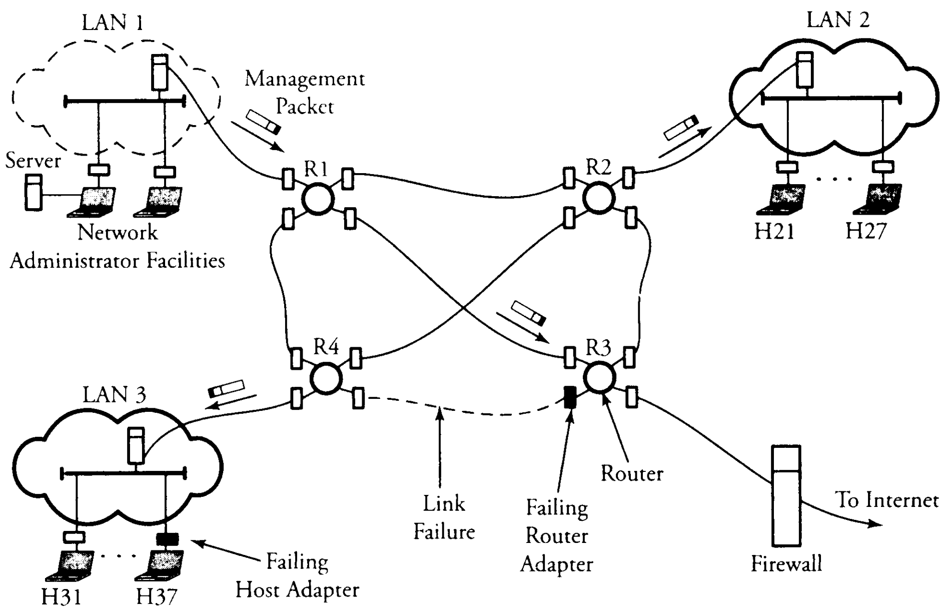

Figure 9.10 Simple network management in a scenario of LANs connecting to the Internet

Network management tasks can be characterized as follows:

- *QoS and performance management.* A network administrator periodically monitors and analyzes routers, hosts, and utilization of links and then redirect traffic flow to avoid any overloaded spots. Certain tools are available to detect rapid changes in traffic flow.
- *Network failure management.* Any fault in a network, such as link, host, or router hardware or software outages, must be detected, located, and responded to by the network. Typically, increased checksum errors in frames is an indication of possible error. Figure 9.10 shows adapter failures at router R3 and host H37; these failures can be detected through network management.
- *Configuration management.* This task involves tracking all the devices under management and ensuring that all devices are connected and operate properly. If there is an unexpected change in routing tables, a network administrator wants to discover the misconfigured spot and reconfigure the network before the error affects the network substantially.
- *Security management.* A network administrator is responsible for the security of its network. This task is handled mainly through firewalls, as discussed in Chapter 10.