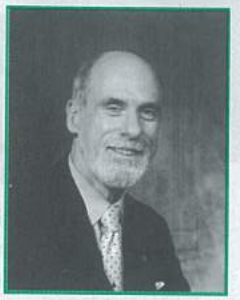
AN INTERVIEW WITH...

Vinton G. Cerf



Vinton G. Cerf is Vice President and Chief Internet Evangelist for Google. He served for over 16 years at MCI in various positions, ending up his tenure there as Senior Vice President for Technology Strategy. He is widely known as the co-designer of the TCP/IP protocols and the architecture of the Internet. During his time from 1976 to 1982 at the US Department of Defense Advanced Research Projects Agency (DARPA), he played a key role leading the development of Internet and Internet-related data packet and security techniques. He received the US Presidential Medal of Freedom in 2005 and the US National Medal of Technology in 1997. He holds a BS in Mathematics from Stanford University and an MS and PhD in computer science from UCLA.

What brought you to specialize in networking?

I was working as a programmer at UCLA in the late 1960s. My job was supported by the US Defense Advanced Research Projects Agency (called ARPA then, called DARPA now). I was working in the laboratory of Professor Leonard Kleinrock on the Network Measurement Center of the newly created ARPAnet. The first node of the ARPAnet was installed at UCLA on September 1, 1969. I was responsible for programming a computer that was used to capture performance information about the ARPAnet and to report this information back for comparison with mathematical models and predictions of the performance of the network.

Several of the other graduate students and I were made responsible for working on the so-called host-level protocols of the ARPAnet—the procedures and formats that would allow many different kinds of computers on the network to interact with each other. It was a fascinating exploration into a new world (for me) of distributed computing and communication.

Did you imagine that IP would become as pervasive as it is today when you first designed the protocol?

When Bob Kahn and I first worked on this in 1973, I think we were mostly very focused on the central question: How can we make heterogeneous packet networks interoperate with one another, assuming we cannot actually change the networks themselves? We hoped that we could find a way to permit an arbitrary collection of packet-switched networks to be interconnected in a transparent fashion, so that host computers could communicate end-to-end without having to do any translations in between. I think we knew that we were dealing

with powerful and expandable technology, but I doubt we had a clear image of what the world would be like with hundreds of millions of computers all interlinked on the Internet.

What do you now envision for the future of networking and the Internet? What major challenges/obstacles do you think lie ahead in their development?

I believe the Internet itself and networks in general will continue to proliferate. Already there is convincing evidence that there will be billions of Internet-enabled devices on the Internet, including appliances like cell phones, refrigerators, personal digital assistants, home servers, televisions, as well as the usual array of laptops, servers, and so on. Big challenges include support for mobility, battery life, capacity of the access links to the network, and ability to scale the optical core of the network up in an unlimited fashion. Designing an interplanetary extension of the Internet is a project in which I am deeply engaged at the Jet Propulsion Laboratory. We will need to cut over from IPv4 [32-bit addresses] to IPv6 [128 bits]. The list is long!

Who has inspired you professionally?

My colleague Bob Kahn; my thesis advisor, Gerald Estrin; my best friend, Steve Crocker (we met in high school and he introduced me to computers in 1960!); and the thousands of engineers who continue to evolve the Internet today.

Do you have any advice for students entering the networking/Internet field?

Think outside the limitations of existing systems—imagine what might be possible; but then do the hard work of figuring out how to get there from the current state of affairs. Dare to dream: A half dozen colleagues and I at the Jet Propulsion Laboratory have been working on the design of an interplanetary extension of the terrestrial Internet. It may take decades to implement this, mission by mission, but to paraphrase: "A man's reach should exceed his grasp, or what are the heavens for?"

CHAPTER 5

The Network Layer: Control Plane

In this chapter, we'll complete our journey through the network layer by covering the **control-plane** component of the network layer—the *network-wide* logic that controls not only how a datagram is forwarded among routers along an end-to-end path from the source host to the destination host, but also how network-layer components and services are configured and managed. In Section 5.2, we'll cover traditional routing algorithms for computing least cost paths in a graph; these algorithms are the basis for two widely deployed Internet routing protocols: OSPF and BGP, that we'll cover in Sections 5.3 and 5.4, respectively. As we'll see, OSPF is a routing protocol that operates within a single ISP's network. BGP is a routing protocol that serves to interconnect all of the networks in the Internet; BGP is thus often referred to as the "glue" that holds the Internet together. Traditionally, control-plane routing protocols have been implemented together with data-plane forwarding functions, monolithically, within a router. As we learned in the introduction to Chapter 4, software-defined networking (SDN) makes a clear separation between the data and control planes, implementing control-plane functions in a separate "controller" service that is distinct, and remote, from the forwarding components of the routers it controls. We'll cover SDN controllers in Section 5.5.

In Sections 5.6 and 5.7 we'll cover some of the nuts and bolts of managing an IP network: ICMP (the Internet Control Message Protocol) and SNMP (the Simple Network Management Protocol).

5.1 Introduction

Let's quickly set the context for our study of the network control plane by recalling Figures 4.2 and 4.3. There, we saw that the forwarding table (in the case of destination-based forwarding) and the flow table (in the case of generalized forwarding) were the principal elements that linked the network layer's data and control planes. We learned that these tables specify the local data-plane forwarding behavior of a router. We saw that in the case of generalized forwarding, the actions taken (Section 4.4.2) could include not only forwarding a packet to a router's output port, but also dropping a packet, replicating a packet, and/or rewriting layer 2, 3 or 4 packet-header fields.

In this chapter, we'll study how those forwarding and flow tables are computed, maintained and installed. In our introduction to the network layer in Section 4.1, we learned that there are two possible approaches for doing so.

- *Per-router control*. Figure 5.1 illustrates the case where a routing algorithm runs in each and every router; both a forwarding and a routing function are contained

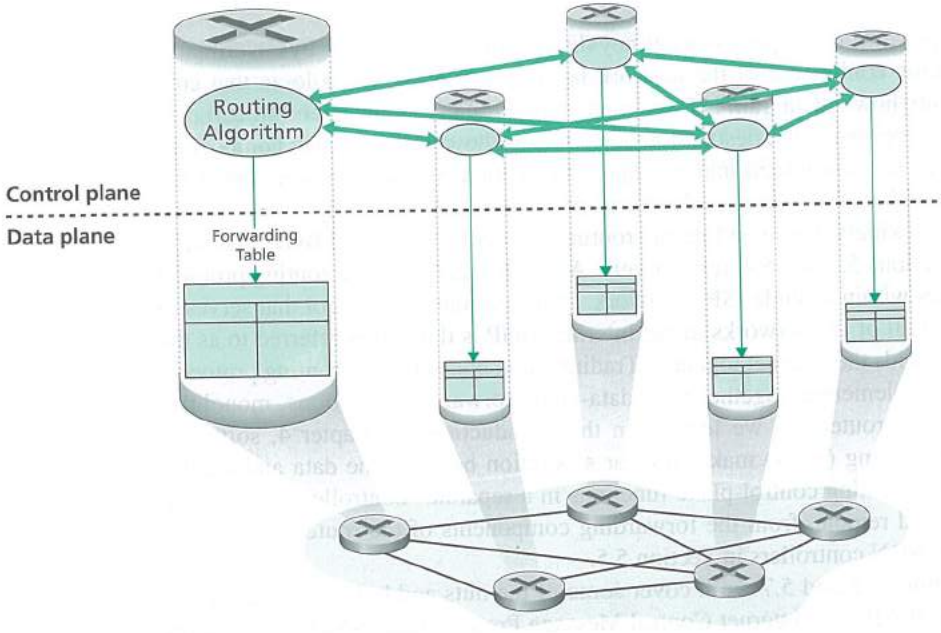


Figure 5.1 ♦ Per-router control: Individual routing algorithm components interact in the control plane

within each router. Each router has a routing component that communicates with the routing components in other routers to compute the values for its forwarding table. This per-router control approach has been used in the Internet for decades. The OSPF and BGP protocols that we'll study in Sections 5.3 and 5.4 are based on this per-router approach to control.

- *Logically centralized control.* Figure 5.2 illustrates the case in which a logically centralized controller computes and distributes the forwarding tables to be used by each and every router. As we saw in Section 4.4, the generalized match-plus-action abstraction allows the router to perform traditional IP forwarding as well as a rich set of other functions (load sharing, firewalling, and NAT) that had been previously implemented in separate middleboxes.

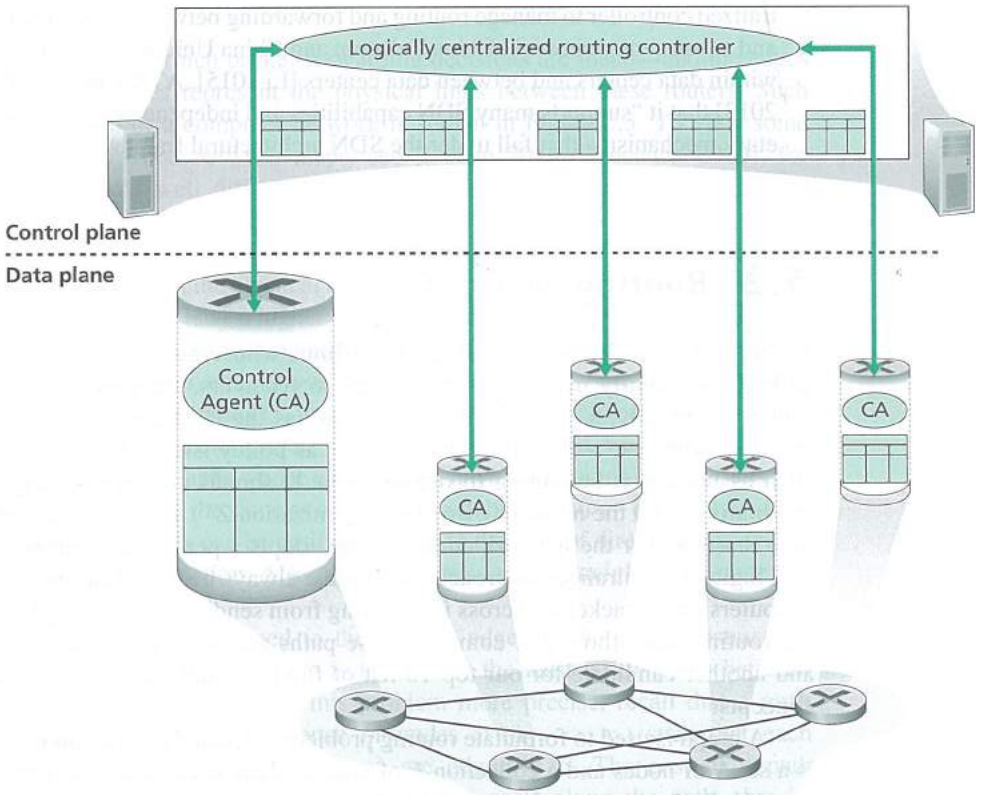


Figure 5.2 ♦ Logically centralized control: A distinct, typically remote, controller interacts with local control agents (CAs)

The controller interacts with a control agent (CA) in each of the routers via a well-defined protocol to configure and manage that router's flow table. Typically, the CA has minimum functionality; its job is to communicate with the controller, and to do as the controller commands. Unlike the routing algorithms in Figure 5.1, the CAs do not directly interact with each other nor do they actively take part in computing the forwarding table. This is a key distinction between per-router control and logically centralized control.

By "logically centralized" control [Levin 2012] we mean that the routing control service is accessed as if it were a single central service point, even though the service is likely to be implemented via multiple servers for fault-tolerance, and performance scalability reasons. As we will see in Section 5.5, SDN adopts this notion of a logically centralized controller—an approach that is finding increased use in production deployments. Google uses SDN to control the routers in its internal B4 global wide-area network that interconnects its data centers [Jain 2013]. SWAN [Hong 2013], from Microsoft Research, uses a logically centralized controller to manage routing and forwarding between a wide area network and a data center network. China Telecom and China Unicom are using SDN both within data centers and between data centers [Li 2015]. AT&T has noted [AT&T 2013] that it "supports many SDN capabilities and independently defined, proprietary mechanisms that fall under the SDN architectural framework."

5.2 Routing Algorithms

In this section we'll study **routing algorithms**, whose goal is to determine good paths (equivalently, routes), from senders to receivers, through the network of routers. Typically, a "good" path is one that has the least cost. We'll see that in practice, however, real-world concerns such as policy issues (for example, a rule such as "router x, belonging to organization Y, should not forward any packets originating from the network owned by organization Z") also come into play. We note that whether the network control plane adopts a per-router control approach or a logically centralized approach, there must always be a well-defined sequence of routers that a packet will cross in traveling from sending to receiving host. Thus, the routing algorithms that compute these paths are of fundamental importance, and another candidate for our top-10 list of fundamentally important networking concepts.

A graph is used to formulate routing problems. Recall that a **graph** $G = (N, E)$ is a set N of nodes and a collection E of edges, where each edge is a pair of nodes from N. In the context of network-layer routing, the nodes in the graph represent

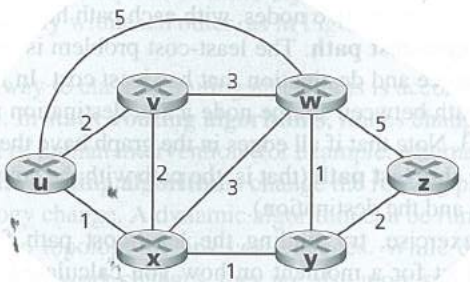


Figure 5.3 ♦ Abstract graph model of a computer network

routers—the points at which packet-forwarding decisions are made—and the edges connecting these nodes represent the physical links between these routers. Such a graph abstraction of a computer network is shown in Figure 5.3. To view some graphs representing real network maps, see [Dodge 2016, Cheswick 2000]; for a discussion of how well different graph-based models model the Internet, see [Zegura 1997, Faloutsos 1999, Li 2004].

As shown in Figure 5.3, an edge also has a value representing its cost. Typically, an edge's cost may reflect the physical length of the corresponding link (for example, a transoceanic link might have a higher cost than a short-haul terrestrial link), the link speed, or the monetary cost associated with a link. For our purposes, we'll simply take the edge costs as a given and won't worry about how they are determined. For any edge (x, y) in E, we denote $c(x, y)$ as the cost of the edge between nodes x and y. If the pair (x, y) does not belong to E, we set $c(x, y) = \infty$. Also, we'll only consider undirected graphs (i.e., graphs whose edges do not have a direction) in our discussion here, so that edge (x, y) is the same as edge (y, x) and that $c(x, y) = c(y, x)$; however, the algorithms we'll study can be easily extended to the case of directed links with a different cost in each direction. Also, a node y is said to be a **neighbor** of node x if (x, y) belongs to E.

Given that costs are assigned to the various edges in the graph abstraction, a natural goal of a routing algorithm is to identify the least costly paths between sources and destinations. To make this problem more precise, recall that a **path** in a graph $G = (N, E)$ is a sequence of nodes $(x_1, x_2, \cdots, x_p)$ such that each of the pairs $(x_1, x_2), (x_2, x_3), \cdots, (x_{p-1}, x_p)$ are edges in E. The cost of a path $(x_1, x_2, \cdots, x_p)$ is simply the sum of all the edge costs along the path, that is,

$c(x_1, x_2) + c(x_2, x_3) + \cdots + c(x_{p-1}, x_p)$. Given any two nodes x and y, there are typically many paths between the two nodes, with each path having a cost. One or more of these paths is a **least-cost path**. The least-cost problem is therefore clear: Find a path between the source and destination that has least cost. In Figure 5.3, for example, the least-cost path between source node u and destination node w is (u, x, y, w) with a path cost of 3. Note that if all edges in the graph have the same cost, the least-cost path is also the **shortest path** (that is, the path with the smallest number of links between the source and the destination).

As a simple exercise, try finding the least-cost path from node u to z in Figure 5.3 and reflect for a moment on how you calculated that path. If you are like most people, you found the path from u to z by examining Figure 5.3, tracing a few routes from u to z, and somehow convincing yourself that the path you had chosen had the least cost among all possible paths. (Did you check all of the 17 possible paths between u and z? Probably not!) Such a calculation is an example of a centralized routing algorithm—the routing algorithm was run in one location, your brain, with complete information about the network. Broadly, one way in which we can classify routing algorithms is according to whether they are centralized or decentralized.

- A **centralized routing algorithm** computes the least-cost path between a source and destination using complete, global knowledge about the network. That is, the algorithm takes the connectivity between all nodes and all link costs as inputs. This then requires that the algorithm somehow obtain this information before actually performing the calculation. The calculation itself can be run at one site (e.g., a logically centralized controller as in Figure 5.2) or could be replicated in the routing component of each and every router (e.g., as in Figure 5.1). The key distinguishing feature here, however, is that the algorithm has complete information about connectivity and link costs. Algorithms with global state information are often referred to as **link-state (LS) algorithms**, since the algorithm must be aware of the cost of each link in the network. We'll study LS algorithms in Section 5.2.1.
- In a **decentralized routing algorithm**, the calculation of the least-cost path is carried out in an iterative, distributed manner by the routers. No node has complete information about the costs of all network links. Instead, each node begins with only the knowledge of the costs of its own directly attached links. Then, through an iterative process of calculation and exchange of information with its neighboring nodes, a node gradually calculates the least-cost path to a destination or set of destinations. The decentralized routing algorithm we'll study below in Section 5.2.2 is called a distance-vector (DV) algorithm, because each node maintains a vector of estimates of the costs (distances) to all other nodes in the network. Such decentralized algorithms, with interactive message exchange between

neighboring routers is perhaps more naturally suited to control planes where the routers interact directly with each other, as in Figure 5.1.

A second broad way to classify routing algorithms is according to whether they are static or dynamic. In **static routing algorithms**, routes change very slowly over time, often as a result of human intervention (for example, a human manually editing a link costs). **Dynamic routing algorithms** change the routing paths as the network traffic loads or topology change. A dynamic algorithm can be run either periodically or in direct response to topology or link cost changes. While dynamic algorithms are more responsive to network changes, they are also more susceptible to problems such as routing loops and route oscillation.

A third way to classify routing algorithms is according to whether they are load-sensitive or load-insensitive. In a **load-sensitive algorithm**, link costs vary dynamically to reflect the current level of congestion in the underlying link. If a high cost is associated with a link that is currently congested, a routing algorithm will tend to choose routes around such a congested link. While early ARPAnet routing algorithms were load-sensitive [McQuillan 1980], a number of difficulties were encountered [Huitema 1998]. Today's Internet routing algorithms (such as RIP, OSPF, and BGP) are **load-insensitive**, as a link's cost does not explicitly reflect its current (or recent past) level of congestion.

5.2.1 The Link-State (LS) Routing Algorithm

Recall that in a link-state algorithm, the network topology and all link costs are known, that is, available as input to the LS algorithm. In practice this is accomplished by having each node broadcast link-state packets to *all* other nodes in the network, with each link-state packet containing the identities and costs of its attached links. In practice (for example, with the Internet's OSPF routing protocol, discussed in Section 5.3) this is often accomplished by a **link-state broadcast** algorithm [Perlman 1999]. The result of the nodes' broadcast is that all nodes have an identical and complete view of the network. Each node can then run the LS algorithm and compute the same set of least-cost paths as every other node.

The link-state routing algorithm we present below is known as *Dijkstra's algorithm*, named after its inventor. A closely related algorithm is Prim's algorithm; see [Cormen 2001] for a general discussion of graph algorithms. Dijkstra's algorithm computes the least-cost path from one node (the source, which we will refer to as u) to all other nodes in the network. Dijkstra's algorithm is iterative and has the property that after the kth iteration of the algorithm, the least-cost paths are known to k destination nodes, and among the least-cost paths to all destination

nodes, these k paths will have the k smallest costs. Let us define the following notation:

- $D(v)$: cost of the least-cost path from the source node to destination v as of this iteration of the algorithm.
- $p(v)$: previous node (neighbor of v) along the current least-cost path from the source to v.
- N': subset of nodes; v is in N' if the least-cost path from the source to v is definitively known.

The centralized routing algorithm consists of an initialization step followed by a loop. The number of times the loop is executed is equal to the number of nodes in the network. Upon termination, the algorithm will have calculated the shortest paths from the source node u to every other node in the network.

Link-State (LS) Algorithm for Source Node u

```
Initialization:
 N' = {u}
 for all nodes v
   if v is a neighbor of u
     then D(v) = c(u,v)
   else D(v) = ∞

Loop
 find w not in N' such that D(w) is a minimum
 add w to N'
 update D(v) for each neighbor v of w and not in N':
        D(v) = min(D(v), D(w)+ c(w,v) )
  /* new cost to v is either old cost to v or known
   least path cost to w plus cost from w to v */
until N'= N
```

As an example, let's consider the network in Figure 5.3 and compute the least-cost paths from u to all possible destinations. A tabular summary of the algorithm's computation is shown in Table 5.1, where each line in the table gives the values of the algorithm's variables at the end of the iteration. Let's consider the few first steps in detail.

- In the initialization step, the currently known least-cost paths from u to its directly attached neighbors, v, x, and w, are initialized to 2, 1, and 5, respectively. Note in

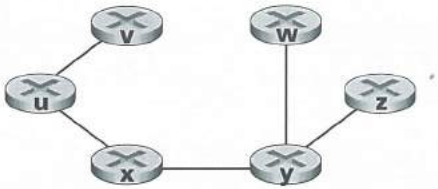
step	N′	D(v), p(v)	D(w), p(w)	D(x), p(x)	D(y), p(y)	D(z), p(z)
0	u	2, u	5, u	1,u	∞	∞
1	ux	2, u	4, x		2, x	∞
2	uxy	2, u	3, y			4, y
3	uxyv		3, y			4, y
4	uxyvw					4, y
5	uxyvwz					

Table 5.1 ♦ Running the link-state algorithm on the network in Figure 5.3

particular that the cost to w is set to 5 (even though we will soon see that a lesser-cost path does indeed exist) since this is the cost of the direct (one hop) link from u to w. The costs to y and z are set to infinity because they are not directly connected to u.

- In the first iteration, we look among those nodes not yet added to the set N' and find that node with the least cost as of the end of the previous iteration. That node is x, with a cost of 1, and thus x is added to the set N'. Line 12 of the LS algorithm is then performed to update $D(v)$ for all nodes v, yielding the results shown in the second line (Step 1) in Table 5.1. The cost of the path to v is unchanged. The cost of the path to w (which was 5 at the end of the initialization) through node x is found to have a cost of 4. Hence this lower-cost path is selected and w's predecessor along the shortest path from u is set to x. Similarly, the cost to y (through x) is computed to be 2, and the table is updated accordingly.
- In the second iteration, nodes v and y are found to have the least-cost paths (2), and we break the tie arbitrarily and add y to the set N' so that N' now contains u, x, and y. The cost to the remaining nodes not yet in N', that is, nodes v, w, and z, are updated via line 12 of the LS algorithm, yielding the results shown in the third row in Table 5.1.
- And so on . . .

When the LS algorithm terminates, we have, for each node, its predecessor along the least-cost path from the source node. For each predecessor, we also have *its* predecessor, and so in this manner we can construct the entire path from the source to all destinations. The forwarding table in a node, say node u, can then be constructed from this information by storing, for each destination, the next-hop node on the least-cost path from u to the destination. Figure 5.4 shows the resulting least-cost paths and forwarding table in u for the network in Figure 5.3.



Destination	Link
v	(u, v)
w	(u, x)
x	(u, x)
y	(u, x)
z	(u, x)

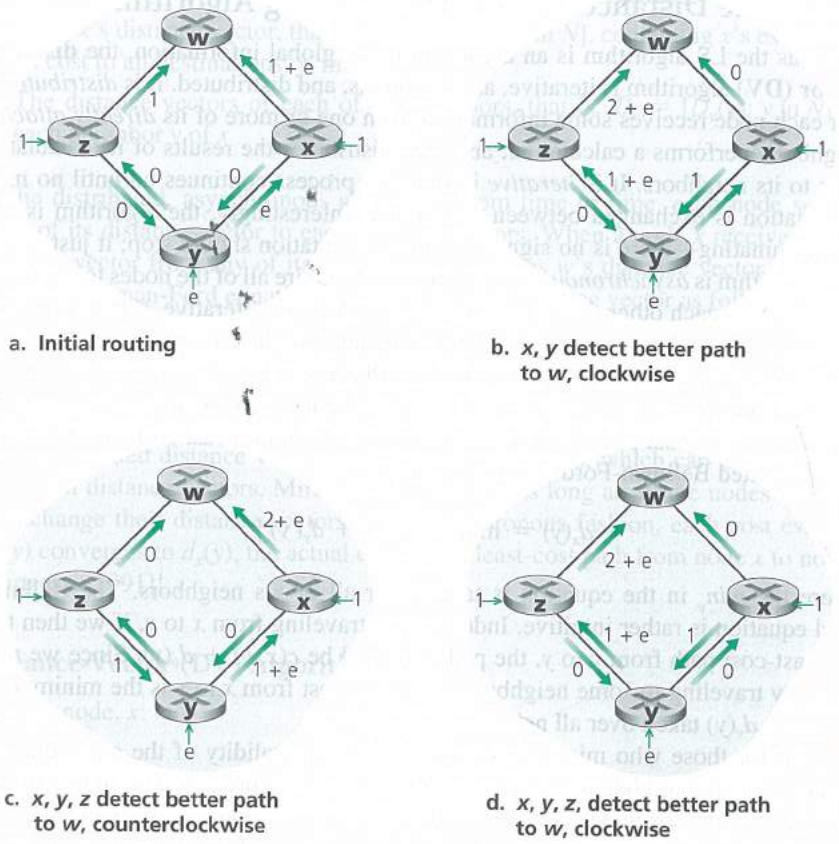
Figure 5.4 ♦ Least cost path and forwarding table for node u

What is the computational complexity of this algorithm? That is, given n nodes (not counting the source), how much computation must be done in the worst case to find the least-cost paths from the source to all destinations? In the first iteration, we need to search through all n nodes to determine the node, w, not in N' that has the minimum cost. In the second iteration, we need to check $n - 1$ nodes to determine the minimum cost; in the third iteration $n - 2$ nodes, and so on. Overall, the total number of nodes we need to search through over all the iterations is $n(n + 1)/2$, and thus we say that the preceding implementation of the LS algorithm has worst-case complexity of order n squared: $O(n^2)$. (A more sophisticated implementation of this algorithm, using a data structure known as a heap, can find the minimum in line 9 in logarithmic rather than linear time, thus reducing the complexity.)

Before completing our discussion of the LS algorithm, let us consider a pathology that can arise. Figure 5.5 shows a simple network topology where link costs are equal to the load carried on the link, for example, reflecting the delay that would be experienced. In this example, link costs are not symmetric; that is, $c(u,v)$ equals $c(v,u)$ only if the load carried on both directions on the link (u,v) is the same. In this example, node z originates a unit of traffic destined for w, node x also originates a unit of traffic destined for w, and node y injects an amount of traffic equal to e, also destined for w. The initial routing is shown in Figure 5.5(a) with the link costs corresponding to the amount of traffic carried.

When the LS algorithm is next run, node y determines (based on the link costs shown in Figure 5.5(a)) that the clockwise path to w has a cost of 1, while the counterclockwise path to w (which it had been using) has a cost of $1 + e$. Hence y's least-cost path to w is now clockwise. Similarly, x determines that its new least-cost path to w is also clockwise, resulting in costs shown in Figure 5.5(b). When the LS algorithm is run next, nodes x, y, and z all detect a zero-cost path to w in the counterclockwise direction, and all route their traffic to the counterclockwise routes. The next time the LS algorithm is run, x, y, and z all then route their traffic to the clockwise routes.

What can be done to prevent such oscillations (which can occur in any algorithm, not just an LS algorithm, that uses a congestion or delay-based link metric)? One solution would be to mandate that link costs not depend on the amount of traffic



a. Initial routing

b. *x, y* detect better path to *w*, clockwise

c. *x, y, z* detect better path to *w*, counterclockwise

d. *x, y, z*, detect better path to *w*, clockwise

Figure 5.5 ♦ Oscillations with congestion-sensitive routing

carried—an unacceptable solution since one goal of routing is to avoid highly congested (for example, high-delay) links. Another solution is to ensure that not all routers run the LS algorithm at the same time. This seems a more reasonable solution, since we would hope that even if routers ran the LS algorithm with the same periodicity, the execution instance of the algorithm would not be the same at each node. Interestingly, researchers have found that routers in the Internet can self-synchronize among themselves [Floyd Synchronization 1994]. That is, even though they initially execute the algorithm with the same period but at different instants of time, the algorithm execution instance can eventually become, and remain, synchronized at the routers. One way to avoid such self-synchronization is for each router to randomize the time it sends out a link advertisement.

Having studied the LS algorithm, let's consider the other major routing algorithm that is used in practice today—the distance-vector routing algorithm.

5.2.2 The Distance-Vector (DV) Routing Algorithm

Whereas the LS algorithm is an algorithm using global information, the **distance-vector (DV)** algorithm is iterative, asynchronous, and distributed. It is *distributed* in that each node receives some information from one or more of its *directly attached* neighbors, performs a calculation, and then distributes the results of its calculation back to its neighbors. It is *iterative* in that this process continues on until no more information is exchanged between neighbors. (Interestingly, the algorithm is also self-terminating—there is no signal that the computation should stop; it just stops.) The algorithm is *asynchronous* in that it does not require all of the nodes to operate in lockstep with each other. We'll see that an asynchronous, iterative, self-terminating, distributed algorithm is much more interesting and fun than a centralized algorithm!

Before we present the DV algorithm, it will prove beneficial to discuss an important relationship that exists among the costs of the least-cost paths. Let $d_x(y)$ be the cost of the least-cost path from node x to node y. Then the least costs are related by the celebrated Bellman-Ford equation, namely,

$$d_x(y) = \min_v \{ c(x, v) + d_v(y) \}, \tag{5.1}$$

where the min_v in the equation is taken over all of x's neighbors. The Bellman-Ford equation is rather intuitive. Indeed, after traveling from x to v, if we then take the least-cost path from v to y, the path cost will be $c(x, v) + d_v(y)$. Since we must begin by traveling to some neighbor v, the least cost from x to y is the minimum of $c(x, v) + d_v(y)$ taken over all neighbors v.

But for those who might be skeptical about the validity of the equation, let's check it for source node u and destination node z in Figure 5.3. The source node u has three neighbors: nodes v, x, and w. By walking along various paths in the graph, it is easy to see that $d_v(z) = 5$, $d_x(z) = 3$, and $d_w(z) = 3$. Plugging these values into Equation 5.1, along with the costs $c(u, v) = 2$, $c(u, x) = 1$, and $c(u, w) = 5$, gives $d_u(z) = \min\{2 + 5, 5 + 3, 1 + 3\} = 4$, which is obviously true and which is exactly what the Dijskstra algorithm gave us for the same network. This quick verification should help relieve any skepticism you may have.

The Bellman-Ford equation is not just an intellectual curiosity. It actually has significant practical importance: the solution to the Bellman-Ford equation provides the entries in node x's forwarding table. To see this, let v^* be any neighboring node that achieves the minimum in Equation 5.1. Then, if node x wants to send a packet to node y along a least-cost path, it should first forward the packet to node v^*. Thus, node x's forwarding table would specify node v^* as the next-hop router for the ultimate destination y. Another important practical contribution of the Bellman-Ford equation is that it suggests the form of the neighbor-to-neighbor communication that will take place in the DV algorithm.

The basic idea is as follows. Each node x begins with $D_x(y)$, an estimate of the cost of the least-cost path from itself to node y, for all nodes, y, in N. Let $\boldsymbol{D}_x = [D_x(y): y \text{ in } N]$ be node x's distance vector, which is the vector of cost estimates from x to all other nodes, y, in N. With the DV algorithm, each node x maintains the following routing information:

- For each neighbor v, the cost $c(x,v)$ from x to directly attached neighbor, v
- Node x's distance vector, that is, $\boldsymbol{D}_x = [D_x(y)\text{: } y \text{ in } N]$, containing x's estimate of its cost to all destinations, y, in N
- The distance vectors of each of its neighbors, that is, $\boldsymbol{D}_v = [D_v(y)\text{: } y \text{ in } N]$ for each neighbor v of x

In the distributed, asynchronous algorithm, from time to time, each node sends a copy of its distance vector to each of its neighbors. When a node x receives a new distance vector from any of its neighbors w, it saves w's distance vector, and then uses the Bellman-Ford equation to update its own distance vector as follows:

$$D_x(y) = \min_v \{ c(x, v) + D_v(y) \} \quad \text{for each node } y \text{ in } N$$

If node x's distance vector has changed as a result of this update step, node x will then send its updated distance vector to each of its neighbors, which can in turn update their own distance vectors. Miraculously enough, as long as all the nodes continue to exchange their distance vectors in an asynchronous fashion, each cost estimate $D_x(y)$ converges to $d_x(y)$, the actual cost of the least-cost path from node x to node y [Bertsekas 1991]!

Distance-Vector (DV) Algorithm

At each node, x:

```
Initialization:
  for all destinations y in N:
     Dx(y)= c(x,y)/* if y is not a neighbor then c(x,y)= ∞ */
  for each neighbor w
     Dw(y) = ? for all destinations y in N
  for each neighbor w
     send distance vector  Dx = [Dx(y): y in N] to w

loop
   wait  (until I see a link cost change to some neighbor w or
            until I receive a distance vector from some neighbor w)

   for each y in N:
       Dx(y) = minv{c(x,v) + Dv(y)}

if Dx(y) changed for any destination y
      send distance vector Dx  = [Dx(y): y in N] to all neighbors

forever
```

In the DV algorithm, a node x updates its distance-vector estimate when it either sees a cost change in one of its directly attached links or receives a distance-vector update from some neighbor. But to update its own forwarding table for a given destination y, what node x really needs to know is not the shortest-path distance to y but instead the neighboring node $v^*(y)$ that is the next-hop router along the shortest path to y. As you might expect, the next-hop router $v^*(y)$ is the neighbor v that achieves the minimum in Line 14 of the DV algorithm. (If there are multiple neighbors v that achieve the minimum, then $v^*(y)$ can be any of the minimizing neighbors.) Thus, in Lines 13–14, for each destination y, node x also determines $v^*(y)$ and updates its forwarding table for destination y.

Recall that the LS algorithm is a centralized algorithm in the sense that it requires each node to first obtain a complete map of the network before running the Dijkstra algorithm. The DV algorithm is *decentralized* and does not use such global information. Indeed, the only information a node will have is the costs of the links to its directly attached neighbors and information it receives from these neighbors. Each node waits for an update from any neighbor (Lines 10–11), calculates its new distance vector when receiving an update (Line 14), and distributes its new distance vector to its neighbors (Lines 16–17). DV-like algorithms are used in many routing protocols in practice, including the Internet's RIP and BGP, ISO IDRP, Novell IPX, and the original ARPAnet.

Figure 5.6 illustrates the operation of the DV algorithm for the simple three-node network shown at the top of the figure. The operation of the algorithm is illustrated in a synchronous manner, where all nodes simultaneously receive distance vectors from their neighbors, compute their new distance vectors, and inform their neighbors if their distance vectors have changed. After studying this example, you should convince yourself that the algorithm operates correctly in an asynchronous manner as well, with node computations and update generation/reception occurring at any time.

The leftmost column of the figure displays three initial **routing tables** for each of the three nodes. For example, the table in the upper-left corner is node x's initial routing table. Within a specific routing table, each row is a distance vector—specifically, each node's routing table includes its own distance vector and that of each of its neighbors. Thus, the first row in node x's initial routing table is $\boldsymbol{D}_x = [D_x(x), D_x(y), D_x(z)] = [0, 2, 7]$. The second and third rows in this table are the most recently received distance vectors from nodes y and z, respectively. Because at initialization node x has not received anything from node y or z, the entries in the second and third rows are initialized to infinity.

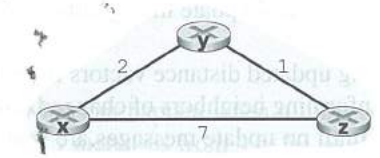
After initialization, each node sends its distance vector to each of its two neighbors. This is illustrated in Figure 5.6 by the arrows from the first column of tables to the second column of tables. For example, node x sends its distance vector $\boldsymbol{D}_x = [0, 2, 7]$ to both nodes y and z. After receiving the updates, each node recomputes its own distance vector. For example, node x computes

$$D_x(x) = 0$$

$$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\} = \min\{2 + 0, 7 + 1\} = 2$$

$$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\} = \min\{2 + 1, 7 + 0\} = 3$$

The second column therefore displays, for each node, the node's new distance vector along with distance vectors just received from its neighbors. Note, for example, that



Node x table

	cost to x	cost to y	cost to z
from x	0	2	7
from y	∞	∞	∞
from z	∞	∞	∞

	cost to x	cost to y	cost to z
from x	0	2	3
from y	2	0	1
from z	7	1	0

	cost to x	cost to y	cost to z
from x	0	2	3
from y	2	0	1
from z	3	1	0

Node y table

	cost to x	cost to y	cost to z
from x	∞	∞	∞
from y	2	0	1
from z	∞	∞	∞

	cost to x	cost to y	cost to z
from x	0	2	7
from y	2	0	1
from z	7	1	0

	cost to x	cost to y	cost to z
from x	0	2	3
from y	2	0	1
from z	3	1	0

Node z table

	cost to x	cost to y	cost to z
from x	∞	∞	∞
from y	∞	∞	∞
from z	7	1	0

	cost to x	cost to y	cost to z
from x	0	2	7
from y	2	0	1
from z	3	1	0

	cost to x	cost to y	cost to z
from x	0	2	3
from y	2	0	1
from z	3	1	0

Time

Figure 5.6 ♦ Distance-vector (DV) algorithm in operation

node x's estimate for the least cost to node z, $D_x(z)$, has changed from 7 to 3. Also note that for node x, neighboring node y achieves the minimum in line 14 of the DV algorithm; thus at this stage of the algorithm, we have at node x that $v^*(y) = y$ and $v^*(z) = y$.

After the nodes recompute their distance vectors, they again send their updated distance vectors to their neighbors (if there has been a change). This is illustrated in Figure 5.6 by the arrows from the second column of tables to the third column of tables. Note that only nodes x and z send updates: node y's distance vector didn't change so node y doesn't send an update. After receiving the updates, the nodes then recompute their distance vectors and update their routing tables, which are shown in the third column.

The process of receiving updated distance vectors from neighbors, recomputing routing table entries, and informing neighbors of changed costs of the least-cost path to a destination continues until no update messages are sent. At this point, since no update messages are sent, no further routing table calculations will occur and the algorithm will enter a quiescent state; that is, all nodes will be performing the wait in Lines 10–11 of the DV algorithm. The algorithm remains in the quiescent state until a link cost changes, as discussed next.

Distance-Vector Algorithm: Link-Cost Changes and Link Failure

When a node running the DV algorithm detects a change in the link cost from itself to a neighbor (Lines 10–11), it updates its distance vector (Lines 13–14) and, if there's a change in the cost of the least-cost path, informs its neighbors (Lines 16–17) of its new distance vector. Figure 5.7(a) illustrates a scenario where the link cost from y to x changes from 4 to 1. We focus here only on y' and z's distance table entries to destination x. The DV algorithm causes the following sequence of events to occur:

- At time t_0, y detects the link-cost change (the cost has changed from 4 to 1), updates its distance vector, and informs its neighbors of this change since its distance vector has changed.
- At time t_1, z receives the update from y and updates its table. It computes a new least cost to x (it has decreased from a cost of 5 to a cost of 2) and sends its new distance vector to its neighbors.
- At time t_2, y receives z's update and updates its distance table. y's least costs do not change and hence y does not send any message to z. The algorithm comes to a quiescent state.

Thus, only two iterations are required for the DV algorithm to reach a quiescent state. The good news about the decreased cost between x and y has propagated quickly through the network.

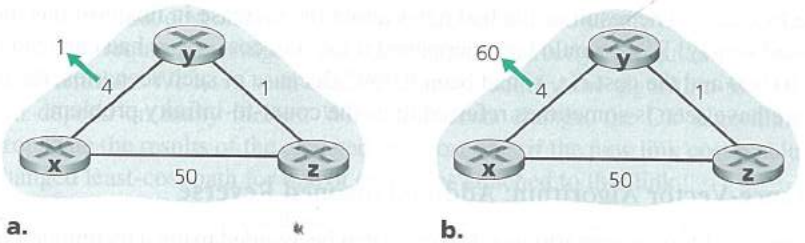


Figure 5.7 ♦ Changes in link cost

Let's now consider what can happen when a link cost *increases*. Suppose that the link cost between x and y increases from 4 to 60, as shown in Figure 5.7(b).

1. Before the link cost changes, $D_y(x) = 4$, $D_y(z) = 1$, $D_z(y) = 1$, and $D_z(x) = 5$. At time t_0, y detects the link-cost change (the cost has changed from 4 to 60). y computes its new minimum-cost path to x to have a cost of

$$D_y(x) = \min\{c(y,x) + D_x(x), c(y,z) + D_z(x)\} = \min\{60 + 0, 1 + 5\} = 6$$

   Of course, with our global view of the network, we can see that this new cost via z is *wrong*. But the only information node y has is that its direct cost to x is 60 and that z has last told y that z could get to x with a cost of 5. So in order to get to x, y would now route through z, fully expecting that z will be able to get to x with a cost of 5. As of t_1 we have a **routing loop**—in order to get to x, y routes through z, and z routes through y. A routing loop is like a black hole—a packet destined for x arriving at y or z as of t_1 will bounce back and forth between these two nodes forever (or until the forwarding tables are changed).
2. Since node y has computed a new minimum cost to x, it informs z of its new distance vector at time t_1.
3. Sometime after t_1, z receives y's new distance vector, which indicates that y's minimum cost to x is 6. z knows it can get to y with a cost of 1 and hence computes a new least cost to x of $D_z(x) = \min\{50 + 0, 1 + 6\} = 7$. Since z's least cost to x has increased, it then informs y of its new distance vector at t_2.
4. In a similar manner, after receiving z's new distance vector, y determines $D_y(x) = 8$ and sends z its distance vector. z then determines $D_z(x) = 9$ and sends y its distance vector, and so on.

How long will the process continue? You should convince yourself that the loop will persist for 44 iterations (message exchanges between y and z)—until z eventually computes the cost of its path via y to be greater than 50. At this point, z will (finally!) determine that its least-cost path to x is via its direct connection to x. y will then

route to x via z. The result of the bad news about the increase in link cost has indeed traveled slowly! What would have happened if the link cost $c(y, x)$ had changed from 4 to 10,000 and the cost $c(z, x)$ had been 9,999? Because of such scenarios, the problem we have seen is sometimes referred to as the count-to-infinity problem.

Distance-Vector Algorithm: Adding Poisoned Reverse

The specific looping scenario just described can be avoided using a technique known as *poisoned reverse*. The idea is simple—if z routes through y to get to destination x, then z will advertise to y that its distance to x is infinity, that is, z will advertise to y that $D_z(x) = \infty$ (even though z knows $D_z(x) = 5$ in truth). z will continue telling this little white lie to y as long as it routes to x via y. Since y believes that z has no path to x, y will never attempt to route to x via z, as long as z continues to route to x via y (and lies about doing so).

Let's now see how poisoned reverse solves the particular looping problem we encountered before in Figure 5.5(b). As a result of the poisoned reverse, y's distance table indicates $D_z(x) = \infty$. When the cost of the (x, y) link changes from 4 to 60 at time t_0, y updates its table and continues to route directly to x, albeit at a higher cost of 60, and informs z of its new cost to x, that is, $D_y(x) = 60$. After receiving the update at t_1, z immediately shifts its route to x to be via the direct (z, x) link at a cost of 50. Since this is a new least-cost path to x, and since the path no longer passes through y, z now informs y that $D_z(x) = 50$ at t_2. After receiving the update from z, y updates its distance table with $D_y(x) = 51$. Also, since z is now on y's least-cost path to x, y poisons the reverse path from z to x by informing z at time t_3 that $D_y(x) = \infty$ (even though y knows that $D_y(x) = 51$ in truth).

Does poisoned reverse solve the general count-to-infinity problem? It does not. You should convince yourself that loops involving three or more nodes (rather than simply two immediately neighboring nodes) will not be detected by the poisoned reverse technique.

A Comparison of LS and DV Routing Algorithms

The DV and LS algorithms take complementary approaches toward computing routing. In the DV algorithm, each node talks to *only* its directly connected neighbors, but it provides its neighbors with least-cost estimates from itself to *all* the nodes (that it knows about) in the network. The LS algorithm requires global information. Consequently, when implemented in each and every router, e.g., as in Figure 4.2 and 5.1, each node would need to communicate with *all* other nodes (via broadcast), but it tells them *only* the costs of its directly connected links. Let's conclude our study of LS and DV algorithms with a quick comparison of some of their attributes. Recall that N is the set of nodes (routers) and E is the set of edges (links).

- *Message complexity.* We have seen that LS requires each node to know the cost of each link in the network. This requires $O(|N|\,|E|)$ messages to be sent. Also,

whenever a link cost changes, the new link cost must be sent to all nodes. The DV algorithm requires message exchanges between directly connected neighbors at each iteration. We have seen that the time needed for the algorithm to converge can depend on many factors. When link costs change, the DV algorithm will propagate the results of the changed link cost only if the new link cost results in a changed least-cost path for one of the nodes attached to that link.

- *Speed of convergence.* We have seen that our implementation of LS is an $O(|N|^2)$ algorithm requiring $O(|N|\,|E|)$ messages. The DV algorithm can converge slowly and can have routing loops while the algorithm is converging. DV also suffers from the count-to-infinity problem.
- *Robustness.* What can happen if a router fails, misbehaves, or is sabotaged? Under LS, a router could broadcast an incorrect cost for one of its attached links (but no others). A node could also corrupt or drop any packets it received as part of an LS broadcast. But an LS node is computing only its own forwarding tables; other nodes are performing similar calculations for themselves. This means route calculations are somewhat separated under LS, providing a degree of robustness. Under DV, a node can advertise incorrect least-cost paths to any or all destinations. (Indeed, in 1997, a malfunctioning router in a small ISP provided national backbone routers with erroneous routing information. This caused other routers to flood the malfunctioning router with traffic and caused large portions of the Internet to become disconnected for up to several hours [Neumann 1997].) More generally, we note that, at each iteration, a node's calculation in DV is passed on to its neighbor and then indirectly to its neighbor's neighbor on the next iteration. In this sense, an incorrect node calculation can be diffused through the entire network under DV.

In the end, neither algorithm is an obvious winner over the other; indeed, both algorithms are used in the Internet.

5.3 Intra-AS Routing in the Internet: OSPF

In our study of routing algorithms so far, we've viewed the network simply as a collection of interconnected routers. One router was indistinguishable from another in the sense that all routers executed the same routing algorithm to compute routing paths through the entire network. In practice, this model and its view of a homogenous set of routers all executing the same routing algorithm is simplistic for two important reasons:

- *Scale.* As the number of routers becomes large, the overhead involved in communicating, computing, and storing routing information becomes prohibitive. Today's Internet consists of hundreds of millions of routers. Storing routing information

for possible destinations at each of these routers would clearly require enormous amounts of memory. The overhead required to broadcast connectivity and link cost updates among all of the routers would be huge! A distance-vector algorithm that iterated among such a large number of routers would surely never converge. Clearly, something must be done to reduce the complexity of route computation in a network as large as the Internet.

- *Administrative autonomy.* As described in Section 1.3, the Internet is a network of ISPs, with each ISP consisting of its own network of routers. An ISP generally desires to operate its network as it pleases (for example, to run whatever routing algorithm it chooses within its network) or to hide aspects of its network's internal organization from the outside. Ideally, an organization should be able to operate and administer its network as it wishes, while still being able to connect its network to other outside networks.

Both of these problems can be solved by organizing routers into **autonomous systems (ASs)**, with each AS consisting of a group of routers that are under the same administrative control. Often the routers in an ISP, and the links that interconnect them, constitute a single AS. Some ISPs, however, partition their network into multiple ASs. In particular, some tier-1 ISPs use one gigantic AS for their entire network, whereas others break up their ISP into tens of interconnected ASs. An autonomous system is identified by its globally unique autonomous system number (ASN) [RFC 1930]. AS numbers, like IP addresses, are assigned by ICANN regional registries [ICANN 2016].

Routers within the same AS all run the same routing algorithm and have information about each other. The routing algorithm running within an autonomous system is called an **intra-autonomous system routing protocol**.

Open Shortest Path First (OSPF)

OSPF routing and its closely related cousin, IS-IS, are widely used for intra-AS routing in the Internet. The Open in OSPF indicates that the routing protocol specification is publicly available (for example, as opposed to Cisco's EIGRP protocol, which was only recently became open [Savage 2015], after roughly 20 years as a Cisco-proprietary protocol). The most recent version of OSPF, version 2, is defined in [RFC 2328], a public document.

OSPF is a link-state protocol that uses flooding of link-state information and a Dijkstra's least-cost path algorithm. With OSPF, each router constructs a complete topological map (that is, a graph) of the entire autonomous system. Each router then locally runs Dijkstra's shortest-path algorithm to determine a shortest-path tree to all *subnets*, with itself as the root node. Individual link costs are configured by the network administrator (see sidebar, Principles and Practice: Setting OSPF Weights). The administrator might choose to set all link costs to 1,

PRINCIPLES IN PRACTICE

SETTING OSPF LINK WEIGHTS

Our discussion of link-state routing has implicitly assumed that link weights are set, a routing algorithm such as OSPF is run, and traffic flows according to the routing tables computed by the LS algorithm. In terms of cause and effect, the link weights are given (i.e., they come first) and result (via Dijkstra's algorithm) in routing paths that minimize overall cost. In this viewpoint, link weights reflect the cost of using a link (e.g., if link weights are inversely proportional to capacity, then the use of high-capacity links would have smaller weight and thus be more attractive from a routing standpoint) and Dijsktra's algorithm serves to minimize overall cost.

In practice, the cause and effect relationship between link weights and routing paths may be reversed, with network operators configuring link weights in order to obtain routing paths that achieve certain traffic engineering goals [Fortz 2000, Fortz 2002]. For example, suppose a network operator has an estimate of traffic flow entering the network at each ingress point and destined for each egress point. The operator may then want to put in place a specific routing of ingress-to-egress flows that minimizes the maximum utilization over all of the network's links. But with a routing algorithm such as OSPF, the operator's main "knobs" for tuning the routing of flows through the network are the link weights. Thus, in order to achieve the goal of minimizing the maximum link utilization, the operator must find the set of link weights that achieves this goal. This is a reversal of the cause and effect relationship—the desired routing of flows is known, and the OSPF link weights must be found such that the OSPF routing algorithm results in this desired routing of flows.

thus achieving minimum-hop routing, or might choose to set the link weights to be inversely proportional to link capacity in order to discourage traffic from using low-bandwidth links. OSPF does not mandate a policy for how link weights are set (that is the job of the network administrator), but instead provides the mechanisms (protocol) for determining least-cost path routing for the given set of link weights.

With OSPF, a router broadcasts routing information to *all* other routers in the autonomous system, not just to its neighboring routers. A router broadcasts link-state information whenever there is a change in a link's state (for example, a change in cost or a change in up/down status). It also broadcasts a link's state periodically (at least once every 30 minutes), even if the link's state has not changed. RFC 2328 notes that "this periodic updating of link state advertisements adds robustness to the link state algorithm." OSPF advertisements are contained in OSPF messages that are

carried directly by IP, with an upper-layer protocol of 89 for OSPF. Thus, the OSPF protocol must itself implement functionality such as reliable message transfer and link-state broadcast. The OSPF protocol also checks that links are operational (via a HELLO message that is sent to an attached neighbor) and allows an OSPF router to obtain a neighboring router's database of network-wide link state.

Some of the advances embodied in OSPF include the following:

- *Security.* Exchanges between OSPF routers (for example, link-state updates) can be authenticated. With authentication, only trusted routers can participate in the OSPF protocol within an AS, thus preventing malicious intruders (or networking students taking their newfound knowledge out for a joyride) from injecting incorrect information into router tables. By default, OSPF packets between routers are not authenticated and could be forged. Two types of authentication can be configured—simple and MD5 (see Chapter 8 for a discussion on MD5 and authentication in general). With simple authentication, the same password is configured on each router. When a router sends an OSPF packet, it includes the password in plaintext. Clearly, simple authentication is not very secure. MD5 authentication is based on shared secret keys that are configured in all the routers. For each OSPF packet that it sends, the router computes the MD5 hash of the content of the OSPF packet appended with the secret key. (See the discussion of message authentication codes in Chapter 8.) Then the router includes the resulting hash value in the OSPF packet. The receiving router, using the preconfigured secret key, will compute an MD5 hash of the packet and compare it with the hash value that the packet carries, thus verifying the packet's authenticity. Sequence numbers are also used with MD5 authentication to protect against replay attacks.
- *Multiple same-cost paths.* When multiple paths to a destination have the same cost, OSPF allows multiple paths to be used (that is, a single path need not be chosen for carrying all traffic when multiple equal-cost paths exist).
- *Integrated support for unicast and multicast routing.* Multicast OSPF (MOSPF) [RFC 1584] provides simple extensions to OSPF to provide for multicast routing. MOSPF uses the existing OSPF link database and adds a new type of link-state advertisement to the existing OSPF link-state broadcast mechanism.
- *Support for hierarchy within a single AS.* An OSPF autonomous system can be configured hierarchically into areas. Each area runs its own OSPF link-state routing algorithm, with each router in an area broadcasting its link state to all other routers in that area. Within each area, one or more area border routers are responsible for routing packets outside the area. Lastly, exactly one OSPF area in the AS is configured to be the backbone area. The primary role of the backbone area is to route traffic between the other areas in the AS. The backbone always contains all area border routers in the AS and may contain non-border routers as well. Inter-area routing within the AS requires that the packet be first

routed to an area border router (intra-area routing), then routed through the backbone to the area border router that is in the destination area, and then routed to the final destination.

OSPF is a relatively complex protocol, and our coverage here has been necessarily brief; [Huitema 1998; Moy 1998; RFC 2328] provide additional details.

5.4 Routing Among the ISPs: BGP

We just learned that OSPF is an example of an intra-AS routing protocol. When routing a packet between a source and destination within the same AS, the route the packet follows is entirely determined by the intra-AS routing protocol. However, to route a packet across multiple ASs, say from a smartphone in Timbuktu to a server in a datacenter in Silicon Valley, we need an **inter-autonomous system routing protocol**. Since an inter-AS routing protocol involves coordination among multiple ASs, communicating ASs must run the same inter-AS routing protocol. In fact, in the Internet, all ASs run the same inter-AS routing protocol, called the Border Gateway Protocol, more commonly known as **BGP** [RFC 4271; Stewart 1999].

BGP is arguably the most important of all the Internet protocols (the only other contender would be the IP protocol that we studied in Section 4.3), as it is the protocol that glues the thousands of ISPs in the Internet together. As we will soon see, BGP is a decentralized and asynchronous protocol in the vein of distance-vector routing described in Section 5.2.2. Although BGP is a complex and challenging protocol, to understand the Internet on a deep level, we need to become familiar with its underpinnings and operation. The time we devote to learning BGP will be well worth the effort.

5.4.1 The Role of BGP

To understand the responsibilities of BGP, consider an AS and an arbitrary router in that AS. Recall that every router has a forwarding table, which plays the central role in the process of forwarding arriving packets to outbound router links. As we have learned, for destinations that are within the same AS, the entries in the router's forwarding table are determined by the AS's intra-AS routing protocol. But what about destinations that are outside of the AS? This is precisely where BGP comes to the rescue.

In BGP, packets are not routed to a specific destination address, but instead to CIDRized prefixes, with each prefix representing a subnet or a collection of subnets. In the world of BGP, a destination may take the form 138.16.68/22, which for this

example includes 1,024 IP addresses. Thus, a router's forwarding table will have entries of the form (x, I), where x is a prefix (such as 138.16.68/22) and I is an interface number for one of the router's interfaces.

As an inter-AS routing protocol, BGP provides each router a means to:

1. *Obtain prefix reachability information from neighboring ASs.* In particular, BGP allows each subnet to advertise its existence to the rest of the Internet. A subnet screams, "I exist and I am here," and BGP makes sure that all the routers in the Internet know about this subnet. If it weren't for BGP, each subnet would be an isolated island—alone, unknown and unreachable by the rest of the Internet.
2. *Determine the "best" routes to the prefixes.* A router may learn about two or more different routes to a specific prefix. To determine the best route, the router will locally run a BGP route-selection procedure (using the prefix reachability information it obtained via neighboring routers). The best route will be determined based on policy as well as the reachability information.

Let us now delve into how BGP carries out these two tasks.

5.4.2 Advertising BGP Route Information

Consider the network shown in Figure 5.8. As we can see, this simple network has three autonomous systems: AS1, AS2, and AS3. As shown, AS3 includes a subnet with prefix x. For each AS, each router is either a **gateway router** or an **internal router**. A gateway router is a router on the edge of an AS that directly connects to one or more routers in other ASs. An **internal router** connects only to hosts and routers within its own AS. In AS1, for example, router 1c is a gateway router; routers 1a, 1b, and 1d are internal routers.

Let's consider the task of advertising reachability information for prefix x to all of the routers shown in Figure 5.8. At a high level, this is straightforward. First, AS3 sends a BGP message to AS2, saying that x exists and is in AS3; let's denote this message as "AS3 x". Then AS2 sends a BGP message to AS1, saying that x exists and that you can get to x by first passing through AS2 and then going to AS3; let's denote that message as "AS2 AS3 x". In this manner, each of the autonomous systems will not only learn about the existence of x, but also learn about a path of autonomous systems that leads to x.

Although the discussion in the above paragraph about advertising BGP reachability information should get the general idea across, it is not precise in the sense that autonomous systems do not actually send messages to each other, but instead routers do. To understand this, let's now re-examine the example in Figure 5.8. In BGP,

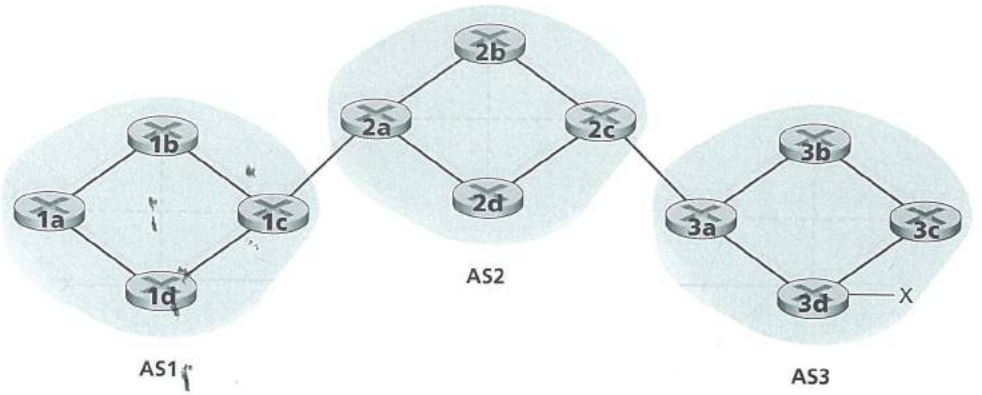


Figure 5.8 ♦ Network with three autonomous systems. AS3 includes a subnet with prefix x

pairs of routers exchange routing information over semi-permanent TCP connections using port 179. Each such TCP connection, along with all the BGP messages sent over the connection, is called a **BGP connection**. Furthermore, a BGP connection that spans two ASs is called an **external BGP (eBGP)** connection, and a BGP session between routers in the same AS is called an **internal BGP (iBGP)** connection. Examples of BGP connections for the network in Figure 5.8 are shown in Figure 5.9. There is typically one eBGP connection for each link that directly connects gateway routers in different ASs; thus, in Figure 5.9, there is an eBGP connection between gateway routers 1c and 2a and an eBGP connection between gateway routers 2c and 3a.

There are also iBGP connections between routers within each of the ASs. In particular, Figure 5.9 displays a common configuration of one BGP connection for each pair of routers internal to an AS, creating a mesh of TCP connections within each AS. In Figure 5.9, the eBGP connections are shown with the long dashes; the iBGP connections are shown with the short dashes. Note that iBGP connections do not always correspond to physical links.

In order to propagate the reachability information, both iBGP and eBGP sessions are used. Consider again advertising the reachability information for prefix x to all routers in AS1 and AS2. In this process, gateway router 3a first sends an eBGP message "AS3 x" to gateway router 2c. Gateway router 2c then sends the iBGP message "AS3 x" to all of the other routers in AS2, including to gateway router 2a. Gateway router 2a then sends the eBGP message "AS2 AS3 x" to gateway router 1c.

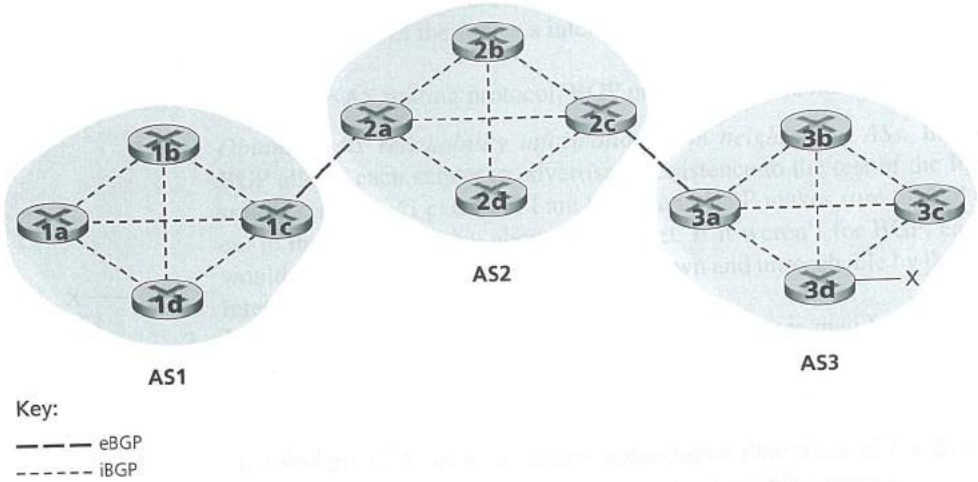


Figure 5.9 ♦ eBGP and iBGP connections

Finally, gateway router 1c uses iBGP to send the message "AS2 AS3 x" to all the routers in AS1. After this process is complete, each router in AS1 and AS2 is aware of the existence of x and is also aware of an AS path that leads to x.

Of course, in a real network, from a given router there may be many different paths to a given destination, each through a different sequence of ASs. For example, consider the network in Figure 5.10, which is the original network in Figure 5.8, with an additional physical link from router 1d to router 3d. In this case, there are two paths from AS1 to x: the path "AS2 AS3 x" via router 1c; and the new path "AS3 x" via the router 1d.

5.4.3 Determining the Best Routes

As we have just learned, there may be many paths from a given router to a destination subnet. In fact, in the Internet, routers often receive reachability information about dozens of different possible paths. How does a router choose among these paths (and then configure its forwarding table accordingly)?

Before addressing this critical question, we need to introduce a little more BGP terminology. When a router advertises a prefix across a BGP connection, it includes with the prefix several **BGP attributes**. In BGP jargon, a prefix along with its attributes is called a **route**. Two of the more important attributes are AS-PATH and NEXT-HOP. The AS-PATH attribute contains the list of ASs through which the

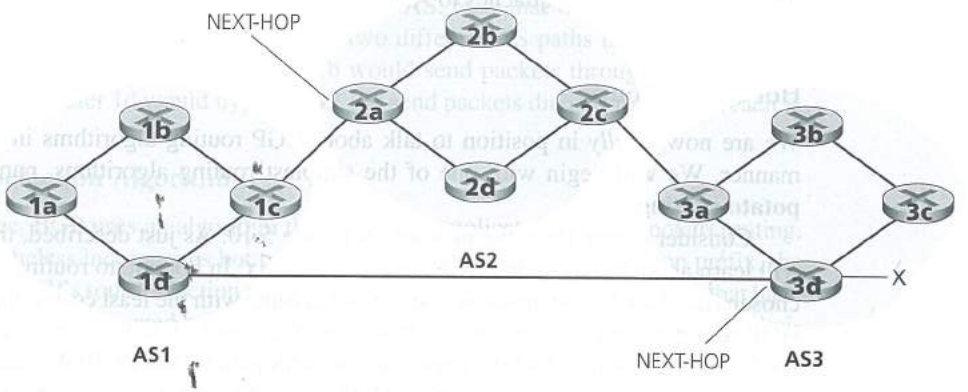


Figure 5.10 ♦ Network augmented with peering link between AS1 and AS3

advertisement has passed, as we've seen in our examples above. To generate the AS-PATH value, when a prefix is passed to an AS, the AS adds its ASN to the existing list in the AS-PATH. For example, in Figure 5.10, there are two routes from AS1 to subnet x: one which uses the AS-PATH "AS2 AS3"; and another that uses the AS-PATH "A3". BGP routers also use the AS-PATH attribute to detect and prevent looping advertisements; specifically, if a router sees that its own AS is contained in the path list, it will reject the advertisement.

Providing the critical link between the inter-AS and intra-AS routing protocols, the NEXT-HOP attribute has a subtle but important use. The NEXT-HOP is the *IP address of the router interface that begins the AS-PATH*. To gain insight into this attribute, let's again refer to Figure 5.10. As indicated in Figure 5.10, the NEXT-HOP attribute for the route "AS2 AS3 x" from AS1 to x that passes through AS2 is the IP address of the left interface on router 2a. The NEXT-HOP attribute for the route "AS3 x" from AS1 to x that bypasses AS2 is the IP address of the leftmost interface of router 3d. In summary, in this toy example, each router in AS1 becomes aware of two BGP routes to prefix x:

IP address of leftmost interface for router 2a; AS2 AS3; x
IP address of leftmost interface of router 3d; AS3; x

Here, each BGP route is written as a list with three components: NEXT-HOP; AS-PATH; destination prefix. In practice, a BGP route includes additional attributes, which we will ignore for the time being. Note that the NEXT-HOP attribute is an IP

address of a router that does *not* belong to AS1; however, the subnet that contains this IP address directly attaches to AS1.

Hot Potato Routing

We are now *finally* in position to talk about BGP routing algorithms in a precise manner. We will begin with one of the simplest routing algorithms, namely, **hot potato routing**.

Consider router 1b in the network in Figure 5.10. As just described, this router will learn about two possible BGP routes to prefix x. In hot potato routing, the route chosen (from among all possible routes) is that route with the least cost to the NEXT-HOP router beginning that route. In this example, router 1b will consult its intra-AS routing information to find the least-cost intra-AS path to NEXT-HOP router 2a and the least-cost intra-AS path to NEXT-HOP router 3d, and then select the route with the smallest of these least-cost paths. For example, suppose that cost is defined as the number of links traversed. Then the least cost from router 1b to router 2a is 2, the least cost from router 1b to router 2d is 3, and router 2a would therefore be selected. Router 1b would then consult its forwarding table (configured by its intra-AS algorithm) and find the interface *I* that is on the least-cost path to router 2a. It then adds (*x*, *I*) to its forwarding table.

The steps for adding an outside-AS prefix in a router's forwarding table for hot potato routing are summarized in Figure 5.11. It is important to note that when adding an outside-AS prefix into a forwarding table, both the inter-AS routing protocol (BGP) and the intra-AS routing protocol (e.g., OSPF) are used.

The idea behind hot-potato routing is for router 1b to get packets out of its AS as quickly as possible (more specifically, with the least cost possible) without worrying about the cost of the remaining portions of the path outside of its AS to the destination. In the name "hot potato routing," a packet is analogous to a hot potato that is burning in your hands. Because it is burning hot, you want to pass it off to another person (another AS) as quickly as possible. Hot potato routing is thus

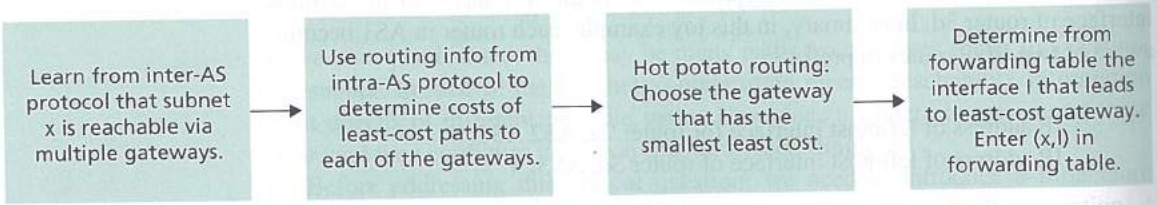


Figure 5.11 ♦ Steps in adding outside-AS destination in a router's forwarding table

a selfish algorithm—it tries to reduce the cost in its own AS while ignoring the other components of the end-to-end costs outside its AS. Note that with hot potato routing, two routers in the same AS may choose two different AS paths to the same prefix. For example, we just saw that router 1b would send packets through AS2 to reach x. However, router 1d would bypass AS2 and send packets directly to AS3 to reach x.

Route-Selection Algorithm

In practice, BGP uses an algorithm that is more complicated than hot potato routing, but nevertheless incorporates hot potato routing. For any given destination prefix, the input into BGP's route-selection algorithm is the set of all routes to that prefix that have been learned and accepted by the router. If there is only one such route, then BGP obviously selects that route. If there are two or more routes to the same prefix, then BGP sequentially invokes the following elimination rules until one route remains:

1. A route is assigned a **local preference** value as one of its attributes (in addition to the AS-PATH and NEXT-HOP attributes). The local preference of a route could have been set by the router or could have been learned from another router in the same AS. The value of the local preference attribute is a policy decision that is left entirely up to the AS's network administrator. (We will shortly discuss BGP policy issues in some detail.) The routes with the highest local preference values are selected.
2. From the remaining routes (all with the same highest local preference value), the route with the shortest AS-PATH is selected. If this rule were the only rule for route selection, then BGP would be using a DV algorithm for path determination, where the distance metric uses the number of AS hops rather than the number of router hops.
3. From the remaining routes (all with the same highest local preference value and the same AS-PATH length), hot potato routing is used, that is, the route with the closest NEXT-HOP router is selected.
4. If more than one route still remains, the router uses BGP identifiers to select the route; see [Stewart 1999].

As an example, let's again consider router 1b in Figure 5.10. Recall that there are exactly two BGP routes to prefix x, one that passes through AS2 and one that bypasses AS2. Also recall that if hot potato routing on its own were used, then BGP would route packets through AS2 to prefix x. But in the above route-selection algorithm, rule 2 is applied before rule 3, causing BGP to select the route that bypasses AS2, since that route has a shorter AS PATH. So we see that with the above route-selection algorithm, BGP is no longer a selfish algorithm—it first looks for routes with short AS paths (thereby likely reducing end-to-end delay).

As noted above, BGP is the *de facto* standard for inter-AS routing for the Internet. To see the contents of various BGP routing tables (large!) extracted from

routers in tier-1 ISPs, see http://www.routeviews.org. BGP routing tables often contain over half a million routes (that is, prefixes and corresponding attributes). Statistics about the size and characteristics of BGP routing tables are presented in [Potaroo 2016].

5.4.4 IP-Anycast

In addition to being the Internet's inter-AS routing protocol, BGP is often used to implement the IP-anycast service [RFC 1546, RFC 7094], which is commonly used in DNS. To motivate IP-anycast, consider that in many applications, we are interested in (1) replicating the same content on different servers in many different dispersed geographical locations, and (2) having each user access the content from the server that is closest. For example, a CDN may replicate videos and other objects on servers in different countries. Similarly, the DNS system can replicate DNS records on DNS servers throughout the world. When a user wants to access this replicated content, it is desirable to point the user to the "nearest" server with the replicated content. BGP's route-selection algorithm provides an easy and natural mechanism for doing so.

To make our discussion concrete, let's describe how a CDN might use IP-anycast. As shown in Figure 5.12, during the IP-anycast configuration stage, the CDN company assigns the *same* IP address to each of its servers, and uses standard BGP to advertise this IP address from each of the servers. When a BGP router receives multiple route advertisements for this IP address, it treats these advertisements as providing different paths to the same physical location (when, in fact, the advertisements are for different paths to different physical locations). When configuring its routing table, each router will locally use the BGP route-selection algorithm to pick the "best" (for example, closest, as determined by AS-hop counts) route to that IP address. For example, if one BGP route (corresponding to one location) is only one AS hop away from the router, and all other BGP routes (corresponding to other locations) are two or more AS hops away, then the BGP router would choose to route packets to the location that is one hop away. After this initial BGP address-advertisement phase, the CDN can do its main job of distributing content. When a client requests the video, the CDN returns to the client the common IP address used by the geographically dispersed servers, no matter where the client is located. When the client sends a request to that IP address, Internet routers then forward the request packet to the "closest" server, as defined by the BGP route-selection algorithm.

Although the above CDN example nicely illustrates how IP-anycast can be used, in practice CDNs generally choose not to use IP-anycast because BGP routing changes can result in different packets of the same TCP connection arriving at different instances of the Web server. But IP-anycast is extensively used by the DNS system to direct DNS queries to the closest root DNS server. Recall from Section 2.4, there are currently 13 IP addresses for root DNS servers. But corresponding

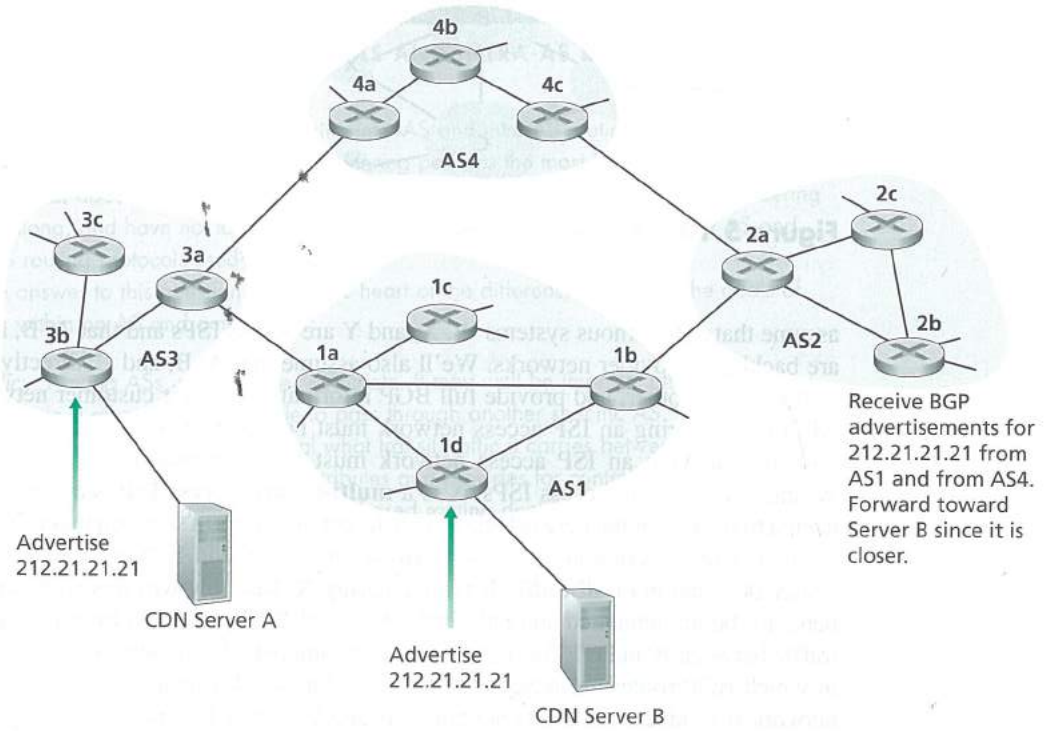


Figure 5.12 ♦ Using IP-anycast to bring users to the closest CDN server

to each of these addresses, there are multiple DNS root servers, with some of these addresses having over 100 DNS root servers scattered over all corners of the world. When a DNS query is sent to one of these 13 IP addresses, IP anycast is used to route the query to the nearest of the DNS root servers that is responsible for that address.

5.4.5 Routing Policy

When a router selects a route to a destination, the AS routing policy can trump all other considerations, such as shortest AS path or hot potato routing. Indeed, in the route-selection algorithm, routes are first selected according to the local-preference attribute, whose value is fixed by the policy of the local AS.

Let's illustrate some of the basic concepts of BGP routing policy with a simple example. Figure 5.13 shows six interconnected autonomous systems: A, B, C, W, X, and Y. It is important to note that A, B, C, W, X, and Y are ASs, not routers. Let's

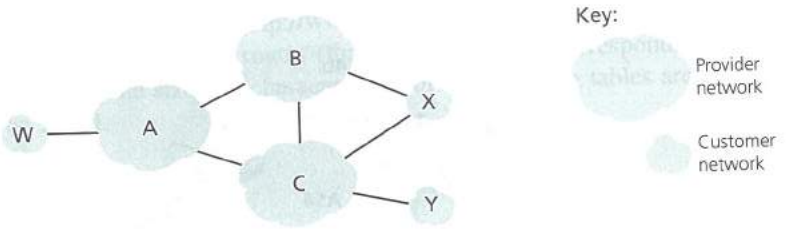


Figure 5.13 ♦ A simple BGP policy scenario

assume that autonomous systems W, X, and Y are access ISPs and that A, B, and C are backbone provider networks. We'll also assume that A, B, and C, directly send traffic to each other, and provide full BGP information to their customer networks. All traffic entering an ISP access network must be destined for that network, and all traffic leaving an ISP access network must have originated in that network. W and Y are clearly access ISPs. X is a **multi-homed access ISP**, since it is connected to the rest of the network via two different providers (a scenario that is becoming increasingly common in practice). However, like W and Y, X itself must be the source/destination of all traffic leaving/entering X. But how will this stub network behavior be implemented and enforced? How will X be prevented from forwarding traffic between B and C? This can easily be accomplished by controlling the manner in which BGP routes are advertised. In particular X will function as an access ISP network if it advertises (to its neighbors B and C) that it has no paths to any other destinations except itself. That is, even though X may know of a path, say XCY, that reaches network Y, it will not advertise this path to B. Since B is unaware that X has a path to Y, B would never forward traffic destined to Y (or C) via X. This simple example illustrates how a selective route advertisement policy can be used to implement customer/provider routing relationships.

Let's next focus on a provider network, say AS B. Suppose that B has learned (from A) that A has a path AW to W. B can thus install the route AW into its routing information base. Clearly, B also wants to advertise the path BAW to its customer, X, so that X knows that it can route to W via B. But should B advertise the path BAW to C? If it does so, then C could route traffic to W via BAW. If A, B, and C are all backbone providers, than B might rightly feel that it should not have to shoulder the burden (and cost!) of carrying transit traffic between A and C. B might rightly feel that it is A's and C's job (and cost!) to make sure that C can route to/from A's customers via a direct connection between A and C. There are currently no official standards that govern how backbone ISPs route among themselves. However, a rule of thumb followed by commercial ISPs is that any traffic flowing across an ISP's backbone network must have either a source or a destination (or both) in a network that is a customer of that ISP; otherwise the traffic would be getting a free ride on the ISP's network. Individual peering agreements (that would govern questions such as

PRINCIPLES IN PRACTICE

WHY ARE THERE DIFFERENT INTER-AS AND INTRA-AS ROUTING PROTOCOLS?

Having now studied the details of specific inter-AS and intra-AS routing protocols deployed in today's Internet, let's conclude by considering perhaps the most fundamental question we could ask about these protocols in the first place (hopefully, you have been wondering this all along, and have not lost the forest for the trees!): Why are different inter-AS and intra-AS routing protocols used?

The answer to this question gets at the heart of the differences between the goals of routing within an AS and among ASs:

- *Policy*. Among ASs, policy issues dominate. It may well be important that traffic originating in a given AS not be able to pass through another specific AS. Similarly, a given AS may well want to control what transit traffic it carries between other ASs. We have seen that BGP carries path attributes and provides for controlled distribution of routing information so that such policy-based routing decisions can be made. Within an AS, everything is nominally under the same administrative control, and thus policy issues play a much less important role in choosing routes within the AS.
- *Scale*. The ability of a routing algorithm and its data structures to scale to handle routing to/among large numbers of networks is a critical issue in inter-AS routing. Within an AS, scalability is less of a concern. For one thing, if a single ISP becomes too large, it is always possible to divide it into two ASs and perform inter-AS routing between the two new ASs. (Recall that OSPF allows such a hierarchy to be built by splitting an AS into areas.)
- *Performance*. Because inter-AS routing is so policy oriented, the quality (for example, performance) of the routes used is often of secondary concern (that is, a longer or more costly route that satisfies certain policy criteria may well be taken over a route that is shorter but does not meet that criteria). Indeed, we saw that among ASs, there is not even the notion of cost (other than AS hop count) associated with routes. Within a single AS, however, such policy concerns are of less importance, allowing routing to focus more on the level of performance realized on a route.

those raised above) are typically negotiated between pairs of ISPs and are often confidential; [Huston 1999a] provides an interesting discussion of peering agreements. For a detailed description of how routing policy reflects commercial relationships among ISPs, see [Gao 2001; Dmitiropoulos 2007]. For a discussion of BGP routing polices from an ISP standpoint, see [Caesar 2005b].

This completes our brief introduction to BGP. Understanding BGP is important because it plays a central role in the Internet. We encourage you to see the references [Griffin 2012; Stewart 1999; Labovitz 1997; Halabi 2000; Huitema 1998; Gao 2001; Feamster 2004; Caesar 2005b; Li 2007] to learn more about BGP.

5.4.6 Putting the Pieces Together: Obtaining Internet Presence

Although this subsection is not about BGP *per se*, it brings together many of the protocols and concepts we've seen thus far, including IP addressing, DNS, and BGP.

Suppose you have just created a small company that has a number of servers, including a public Web server that describes your company's products and services, a mail server from which your employees obtain their e-mail messages, and a DNS server. Naturally, you would like the entire world to be able to visit your Web site in order to learn about your exciting products and services. Moreover, you would like your employees to be able to send and receive e-mail to potential customers throughout the world.

To meet these goals, you first need to obtain Internet connectivity, which is done by contracting with, and connecting to, a local ISP. Your company will have a gateway router, which will be connected to a router in your local ISP. This connection might be a DSL connection through the existing telephone infrastructure, a leased line to the ISP's router, or one of the many other access solutions described in Chapter 1. Your local ISP will also provide you with an IP address range, e.g., a /24 address range consisting of 256 addresses. Once you have your physical connectivity and your IP address range, you will assign one of the IP addresses (in your address range) to your Web server, one to your mail server, one to your DNS server, one to your gateway router, and other IP addresses to other servers and networking devices in your company's network.

In addition to contracting with an ISP, you will also need to contract with an Internet registrar to obtain a domain name for your company, as described in Chapter 2. For example, if your company's name is, say, Xanadu Inc., you will naturally try to obtain the domain name xanadu.com. Your company must also obtain presence in the DNS system. Specifically, because outsiders will want to contact your DNS server to obtain the IP addresses of your servers, you will also need to provide your registrar with the IP address of your DNS server. Your registrar will then put an entry for your DNS server (domain name and corresponding IP address) in the .com top-level-domain servers, as described in Chapter 2. After this step is completed, any user who knows your domain name (e.g., xanadu.com) will be able to obtain the IP address of your DNS server via the DNS system.

So that people can discover the IP addresses of your Web server, in your DNS server you will need to include entries that map the host name of your Web server (e.g., www.xanadu.com) to its IP address. You will want to have similar entries for

other publicly available servers in your company, including your mail server. In this manner, if Alice wants to browse your Web server, the DNS system will contact your DNS server, find the IP address of your Web server, and give it to Alice. Alice can then establish a TCP connection directly with your Web server.

However, there still remains one other necessary and crucial step to allow outsiders from around the world to access your Web server. Consider what happens when Alice, who knows the IP address of your Web server, sends an IP datagram (e.g., a TCP SYN segment) to that IP address. This datagram will be routed through the Internet, visiting a series of routers in many different ASs, and eventually reach your Web server. When any one of the routers receives the datagram, it is going to look for an entry in its forwarding table to determine on which outgoing port it should forward the datagram. Therefore, each of the routers needs to know about the existence of your company's /24 prefix (or some aggregate entry). How does a router become aware of your company's prefix? As we have just seen, it becomes aware of it from BGP! Specifically, when your company contracts with a local ISP and gets assigned a prefix (i.e., an address range), your local ISP will use BGP to advertise your prefix to the ISPs to which it connects. Those ISPs will then, in turn, use BGP to propagate the advertisement. Eventually, all Internet routers will know about your prefix (or about some aggregate that includes your prefix) and thus be able to appropriately forward datagrams destined to your Web and mail servers.

5.5 The SDN Control Plane

In this section, we'll dive into the SDN control plane—the network-wide logic that controls packet forwarding among a network's SDN-enabled devices, as well as the configuration and management of these devices and their services. Our study here builds on our earlier discussion of generalized SDN forwarding in Section 4.4, so you might want to first review that section, as well as Section 5.1 of this chapter, before continuing on. As in Section 4.4, we'll again adopt the terminology used in the SDN literature and refer to the network's forwarding devices as "packet switches" (or just switches, with "packet" being understood), since forwarding decisions can be made on the basis of network-layer source/destination addresses, link-layer source/destination addresses, as well as many other values in transport-, network-, and link-layer packet-header fields.

Four key characteristics of an SDN architecture can be identified [Kreutz 2015]:

- *Flow-based forwarding*. Packet forwarding by SDN-controlled switches can be based on any number of header field values in the transport-layer, network-layer, or link-layer header. We saw in Section 4.4 that the OpenFlow1.0 abstraction allows forwarding based on eleven different header field values. This contrasts

sharply with the traditional approach to router-based forwarding that we studied in Sections 5.2–5.4, where forwarding of IP datagrams was based solely on a datagram's destination IP address. Recall from Figure 5.2 that packet forwarding rules are specified in a switch's flow table; it is the job of the SDN control plane to compute, manage and install flow table entries in all of the network's switches.

- *Separation of data plane and control plane.* This separation is shown clearly in Figures 5.2 and 5.14. The data plane consists of the network's switches—relatively simple (but fast) devices that execute the "match plus action" rules in their flow tables. The control plane consists of servers and software that determine and manage the switches' flow tables.
- *Network control functions: external to data-plane switches.* Given that the "S" in SDN is for "software," it's perhaps not surprising that the SDN control plane is implemented in software. Unlike traditional routers, however, this software executes on servers that are both distinct and remote from the network's switches. As shown in Figure 5.14, the control plane itself consists of two components—an SDN controller (or network operating system [Gude 2008]) and a set of network-control applications. The controller maintains accurate network state information (e.g., the state of remote links, switches, and hosts); provides this information to the network-control applications running in the control plane; and provides the means through which these applications can monitor, program, and control the underlying network devices. Although the controller in Figure 5.14 is shown as a single central server, in practice the controller is only logically centralized; it is typically implemented on several servers that provide coordinated, scalable performance and high availability.
- *A programmable network.* The network is programmable through the network-control applications running in the control plane. These applications represent the "brains" of the SDN control plane, using the APIs provided by the SDN controller to specify and control the data plane in the network devices. For example, a routing network-control application might determine the end-end paths between sources and destinations (e.g., by executing Dijkstra's algorithm using the node-state and link-state information maintained by the SDN controller). Another network application might perform access control, i.e., determine which packets are to be blocked at a switch, as in our third example in Section 4.4.3. Yet another application might forward packets in a manner that performs server load balancing (the second example we considered in Section 4.4.3).

From this discussion, we can see that SDN represents a significant "unbundling" of network functionality—data plane switches, SDN controllers, and network-control applications are separate entities that may each be provided by different vendors and organizations. This contrasts with the pre-SDN model in which a switch/router (together with its embedded control plane software and protocol implementations) was monolithic, vertically integrated, and sold by a single vendor. This unbundling

of network functionality in SDN has been likened to the earlier evolution from mainframe computers (where hardware, system software, and applications were provided by a single vendor) to personal computers (with their separate hardware, operating systems, and applications). The unbundling of computing hardware, system software, and applications has arguably led to a rich, open ecosystem driven by innovation in all three of these areas; one hope for SDN is that it too will lead to a such rich innovation.

Given our understanding of the SDN architecture of Figure 5.14, many questions naturally arise. How and where are the flow tables actually computed? How are these tables updated in response to events at SDN-controlled devices (e.g., an attached link going up/down)? And how are the flow table entries at multiple switches coordinated in such a way as to result in orchestrated and consistent network-wide functionality (e.g., end-to-end paths for forwarding packets from sources to destinations, or coordinated distributed firewalls)? It is the role of the SDN control plane to provide these, and many other, capabilities.

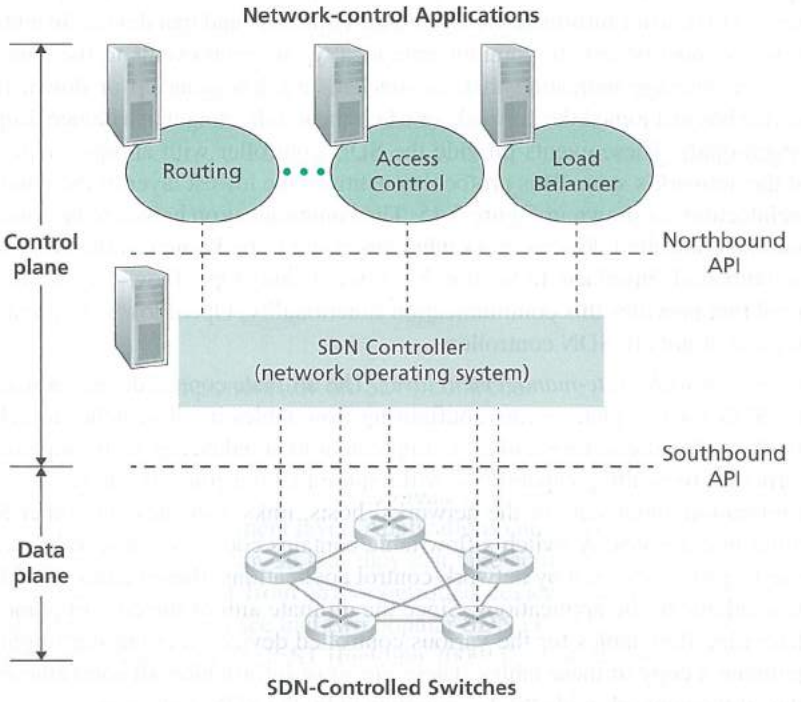


Figure 5.14 ♦ Components of the SDN architecture: SDN-controlled switches, the SDN controller, network-control applications

5.5.2 The SDN Control Plane: SDN Controller and SDN Network-control Applications

Let's begin our discussion of the SDN control plane in the abstract, by considering the generic capabilities that the control plane must provide. As we'll see, this abstract, "first principles" approach will lead us to an overall architecture that reflects how SDN control planes have been implemented in practice.

As noted above, the SDN control plane divides broadly into two components—the SDN controller and the SDN network-control applications. Let's explore the controller first. Many SDN controllers have been developed since the earliest SDN controller [Gude 2008]; see [Kreutz 2015] for an extremely thorough and up-to-date survey. Figure 5.15 provides a more detailed view of a generic SDN controller. A controller's functionality can be broadly organized into three layers. Let's consider these layers in an uncharacteristically bottom-up fashion:

- *A communication layer: communicating between the SDN controller and controlled network devices.* Clearly, if an SDN controller is going to control the operation of a remote SDN-enabled switch, host, or other device, a protocol is needed to transfer information between the controller and that device. In addition, a device must be able to communicate locally-observed events to the controller (e.g., a message indicating that an attached link has gone up or down, that a device has just joined the network, or a heartbeat indicating that a device is up and operational). These events provide the SDN controller with an up-to-date view of the network's state. This protocol constitutes the lowest layer of the controller architecture, as shown in Figure 5.15. The communication between the controller and the controlled devices cross what has come to be known as the controller's "southbound" interface. In Section 5.5.2, we'll study OpenFlow—a specific protocol that provides this communication functionality. OpenFlow is implemented in most, if not all, SDN controllers.
- *A network-wide state-management layer.* The ultimate control decisions made by the SDN control plane—e.g., configuring flow tables in all switches to achieve the desired end-end forwarding, to implement load balancing, or to implement a particular firewalling capability—will require that the controller have up-to-date information about state of the networks' hosts, links, switches, and other SDN-controlled devices. A switch's flow table contains counters whose values might also be profitably used by network-control applications; these values should thus be available to the applications. Since the ultimate aim of the control plane is to determine flow tables for the various controlled devices, a controller might also maintain a copy of these tables. These pieces of information all constitute examples of the network-wide "state" maintained by the SDN controller.
- *The interface to the network-control application layer.* The controller interacts with network-control applications through its "northbound" interface. This API

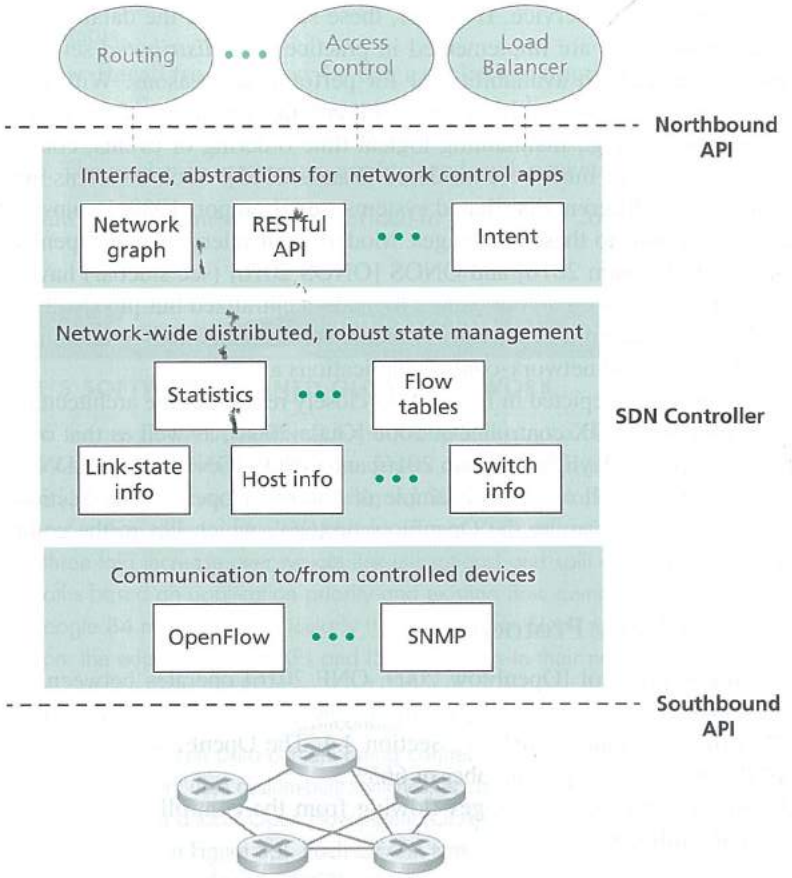


Figure 5.15 ♦ Components of an SDN controller

allows network-control applications to read/write network state and flow tables within the state-management layer. Applications can register to be notified when state-change events occur, so that they can take actions in response to network event notifications sent from SDN-controlled devices. Different types of APIs may be provided; we'll see that two popular SDN controllers communicate with their applications using a REST [Fielding 2000] request-response interface.

We have noted several times that an SDN controller can be considered to be "logically centralized," i.e., that the controller may be viewed externally (e.g., from the point of view of SDN-controlled devices and external network-control applications)

as a single, monolithic service. However, these services and the databases used to hold state information are implemented in practice by a *distributed* set of servers for fault tolerance, high availability, or for performance reasons. With controller functions being implemented by a *set* of servers, the semantics of the controller's internal operations (e.g., maintaining logical time ordering of events, consistency, consensus, and more) must be considered [Panda 2013]. Such concerns are common across many different distributed systems; see [Lamport 1989, Lampson 1996] for elegant solutions to these challenges. Modern controllers such as OpenDaylight [OpenDaylight Lithium 2016] and ONOS [ONOS 2016] (see sidebar) have placed considerable emphasis on architecting a logically centralized but physically distributed controller platform that provides scalable services and high availability to the controlled devices and network-control applications alike.

The architecture depicted in Figure 5.15 closely resembles the architecture of the originally proposed NOX controller in 2008 [Gude 2008], as well as that of today's OpenDaylight [OpenDaylight Lithium 2016] and ONOS [ONOS 2016] SDN controllers (see sidebar). We'll cover an example of controller operation in Section 5.5.3. First, however, let's examine the OpenFlow protocol, which lies in the controller's communication layer.

5.5.2 OpenFlow Protocol

The OpenFlow protocol [OpenFlow 2009, ONF 2016] operates between an SDN controller and an SDN-controlled switch or other device implementing the OpenFlow API that we studied earlier in Section 4.4. The OpenFlow protocol operates over TCP, with a default port number of 6653.

Among the important messages flowing from the controller to the controlled switch are the following:

- *Configuration.* This message allows the controller to query and set a switch's configuration parameters.
- *Modify-State.* This message is used by a controller to add/delete or modify entries in the switch's flow table, and to set switch port properties.
- *Read-State.* This message is used by a controller to collect statistics and counter values from the switch's flow table and ports.
- *Send-Packet.* This message is used by the controller to send a specific packet out of a specified port at the controlled switch. The message itself contains the packet to be sent in its payload.

Among the messages flowing from the SDN-controlled switch to the controller are the following:

- *Flow-Removed.* This message informs the controller that a flow table entry has been removed, for example by a timeout or as the result of a received *modify-state* message.

- *Port-status.* This message is used by a switch to inform the controller of a change in port status.
- *Packet-in.* Recall from Section 4.4 that a packet arriving at a switch port and not matching any flow table entry is sent to the controller for additional processing. Matched packets may also be sent to the controller, as an action to be taken on a match. The *packet-in* message is used to send such packets to the controller.

Additional OpenFlow messages are defined in [OpenFlow 2009, ONF 2016].

PRINCIPLES IN PRACTICE

GOOGLE'S SOFTWARE-DEFINED GLOBAL NETWORK

Recall from the case study in Section 2.6 that Google deploys a dedicated wide-area network (WAN) that interconnects its data centers and server clusters (in IXPs and ISPs). This network, called B4, has a Google-designed SDN control plane built on OpenFlow. Google's network is able to drive WAN links at near 70% utilization over the long run (a two to three fold increase over typical link utilizations) and split application flows among multiple paths based on application priority and existing flow demands [Jain 2013].

The Google B4 network is particularly it well-suited for SDN: *(i)* Google controls all devices from the edge servers in IXPs and ISPs to routers in their network core; *(ii)* the most bandwidth-intensive applications are large-scale data copies between sites that can defer to higher-priority interactive applications during times of resource congestion; *(iii)* with only a few dozen data centers being connected, centralized control is feasible.

Google's B4 network uses custom-built switches, each implementing a slightly extended version of OpenFlow, with a local Open Flow Agent (OFA) that is similar in spirit to the control agent we encountered in Figure 5.2. Each OFA in turn connects to an Open Flow Controller (OFC) in the network control server (NCS), using a separate "out of band" network, distinct from the network that carries data-center traffic between data centers. The OFC thus provides the services used by the NCS to communicate with its controlled switches, similar in spirit to the lowest layer in the SDN architecture shown in Figure 5.15. In B4, the OFC also performs state management functions, keeping node and link status in a Network Information Base (NIB). Google's implementation of the OFC is based on the ONIX SDN controller [Koponen 2010]. Two routing protocols, BGP (for routing between the data centers) and IS-IS (a close relative of OSPF, for routing within a data center), are implemented. Paxos [Chandra 2007] is used to execute hot replicas of NCS components to protect against failure.

A traffic engineering network-control application, sitting logically above the set of network control servers, interacts with these servers to provide global, network-wide bandwidth provisioning for groups of application flows. With B4, SDN made an important leap forward into the operational networks of a global network provider. See [Jain 2013] for a detailed description of B4.

5.5.3 Data and Control Plane Interaction: An Example

In order to solidify our understanding of the interaction between SDN-controlled switches and the SDN controller, let's consider the example shown in Figure 5.16, in which Dijkstra's algorithm (which we studied in Section 5.2) is used to determine shortest path routes. The SDN scenario in Figure 5.16 has two important differences from the earlier per-router-control scenario of Sections 5.2.1 and 5.3, where Dijkstra's algorithm was implemented in each and every router and link-state updates were flooded among all network routers:

- Dijkstra's algorithm is executed as a separate application, outside of the packet switches.
- Packet switches send link updates to the SDN controller and not to each other.

In this example, let's assume that the link between switch s1 and s2 goes down; that shortest path routing is implemented, and consequently and that incoming and outgoing flow forwarding rules at s1, s3, and s4 are affected, but that s2's

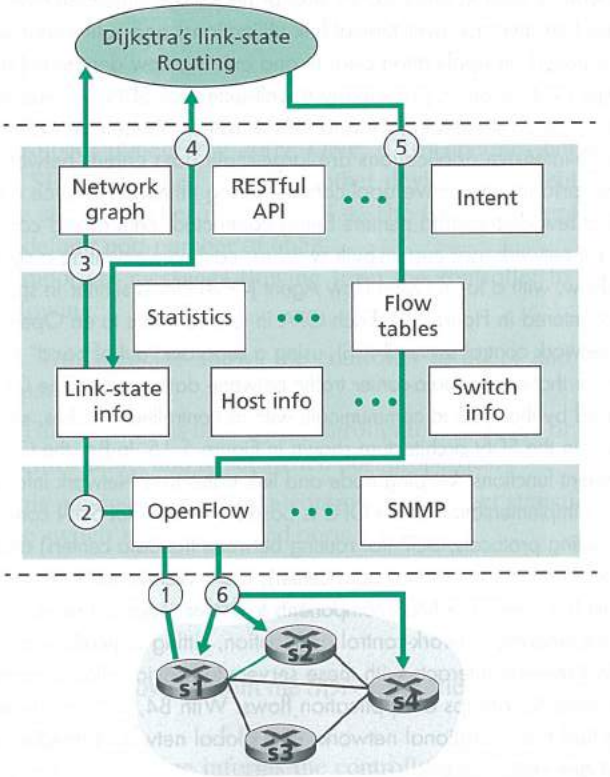


Figure 5.16 ♦ SDN controller scenario: Link-state change

operation is unchanged. Let's also assume that OpenFlow is used as the communication layer protocol, and that the control plane performs no other function other than link-state routing.

1. Switch s1, experiencing a link failure between itself and s2, notifies the SDN controller of the link-state change using the OpenFlow *port-status* message.
2. The SDN controller receives the OpenFlow message indicating the link-state change, and notifies the link-state manager, which updates a link-state database.
3. The network-control application that implements Dijkstra's link-state routing has previously registered to be notified when link state changes. That application receives the notification of the link-state change.
4. The link-state routing application interacts with the link-state manager to get updated link state; it might also consult other components in the state-management layer. It then computes the new least-cost paths.
5. The link-state routing application then interacts with the flow table manager, which determines the flow tables to be updated.
6. The flow table manager then uses the OpenFlow protocol to update flow table entries at affected switches—s1 (which will now route packets destined to s2 via s4), s2 (which will now begin receiving packets from s1 via intermediate switch s4), and s4 (which must now forward packets from s1 destined to s2).

This example is simple but illustrates how the SDN control plane provides control-plane services (in this case network-layer routing) that had been previously implemented with per-router control exercised in each and every network router. One can now easily appreciate how an SDN-enabled ISP could easily switch from least-cost path routing to a more hand-tailored approach to routing. Indeed, since the controller can tailor the flow tables as it pleases, it can implement *any* form of forwarding that it pleases—simply by changing its application-control software. This ease of change should be contrasted to the case of a traditional per-router control plane, where software in all routers (which might be provided to the ISP by multiple independent vendors) must be changed.

5.5.4 SDN: Past and Future

Although the intense interest in SDN is a relatively recent phenomenon, the technical roots of SDN, and the separation of the data and control planes in particular, go back considerably further. In 2004, [Feamster 2004, Lakshman 2004, RFC 3746] all argued for the separation of the network's data and control planes. [van der Merwe 1998] describes a control framework for ATM networks [Black 1995] with multiple controllers, each controlling a number of ATM switches. The Ethane project [Casado 2007] pioneered the notion of a network of simple flow-based Ethernet switches with match-plus-action flow tables, a centralized controller that managed flow admission

and routing, and the forwarding of unmatched packets from the switch to the controller. A network of more than 300 Ethane switches was operational in 2007. Ethane quickly evolved into the OpenFlow project, and the rest (as the saying goes) is history!

Numerous research efforts are aimed at developing future SDN architectures and capabilities. As we have seen, the SDN revolution is leading to the disruptive replacement of dedicated monolithic switches and routers (with both data and control planes) by simple commodity switching hardware and a sophisticated software control plane. A generalization of SDN known as network functions virtualization (NFV) similarly aims at disruptive replacement of sophisticated middleboxes (such as middleboxes with dedicated hardware and proprietary software for media caching/service) with simple commodity servers, switching, and storage [Gember-Jacobson 2014]. A second area of important research seeks to extend SDN concepts from the intra-AS setting to the inter-AS setting [Gupta 2014].

PRINCIPLES IN PRACTICE

SDN CONTROLLER CASE STUDIES: THE OPENDAYLIGHT AND ONOS CONTROLLERS

In the earliest days of SDN, there was a single SDN protocol (OpenFlow [McKeown 2008; OpenFlow 2009]) and a single SDN controller (NOX [Gude 2008]). Since then, the number of SDN controllers in particular has grown significantly [Kreutz 2015]. Some SDN controllers are company-specific and proprietary, e.g., ONIX [Koponen 2010], Juniper Networks Contrail [Juniper Contrail 2016], and Google's controller [Jain 2013] for its B4 wide-area network. But many more controllers are open-source and implemented in a variety of programming languages [Erickson 2013]. Most recently, the OpenDaylight controller [OpenDaylight Lithium 2016] and the ONOS controller [ONOS 2016] have found considerable industry support. They are both open-source and are being developed in partnership with the Linux Foundation.

The OpenDaylight Controller

Figure 5.17 presents a simplified view of the OpenDaylight Lithium SDN controller platform [OpenDaylight Lithium 2016]. ODL's main set of controller components correspond closely to those we developed in Figure 5.15.

Network-Service Applications are the applications that determine how data-plane forwarding and other services, such as firewalling and load balancing, are accomplished in the controlled switches. Unlike the canonical controller in Figure 5.15, the ODL controller has *two* interfaces through which applications may communicate with native controller services and each other: external applications communicate with controller modules using a REST request-response API running over HTTP. Internal applications communicate with each other via the Service Abstraction Layer (SAL). The choice as to whether a controller application is implemented externally or internally is up to the application designer;

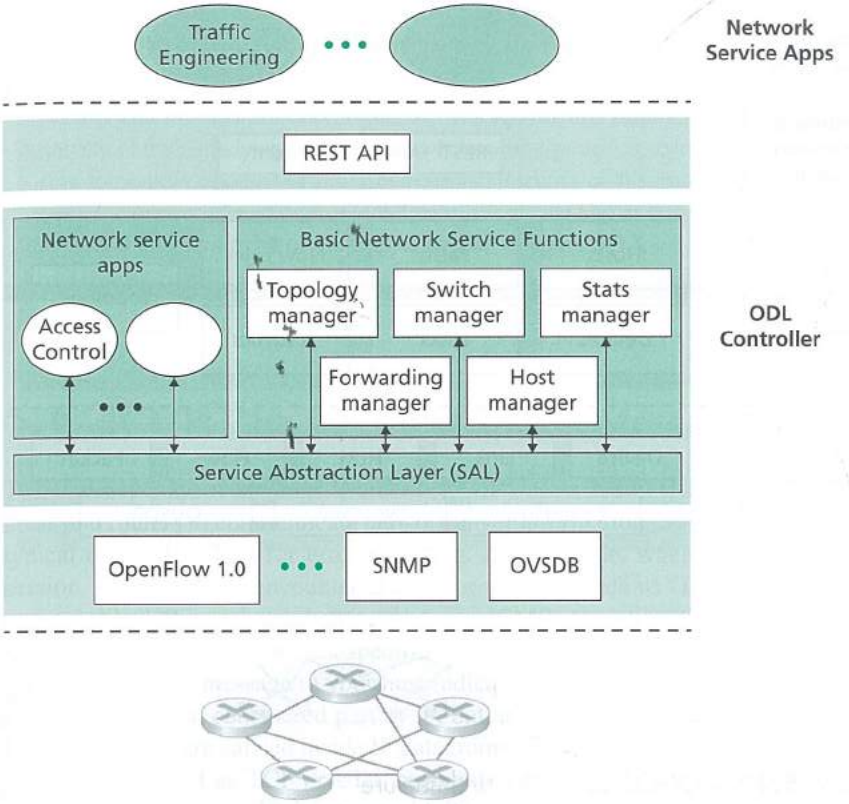


Figure 5.17 ♦ The OpenDaylight controller

the particular configuration of applications shown in Figure 5.17 is only meant as an example.

ODL's *Basic Network-Service Functions* are at the heart of the controller, and they correspond closely to the network-wide state management capabilities that we encountered in Figure 5.15. The SAL is the controller's nerve center, allowing controller components and applications to invoke each other's services and to subscribe to events they generate. It also provides a uniform abstract interface to the specific *underlying communications protocols* in the communication layer, including OpenFlow and SNMP (the Simple Network Management Protocol—a network management protocol that we will cover in Section 5.7). OVSDB is a protocol used to manage data center switching, an important application area for SDN technology. We'll introduce data center networking in Chapter 6.

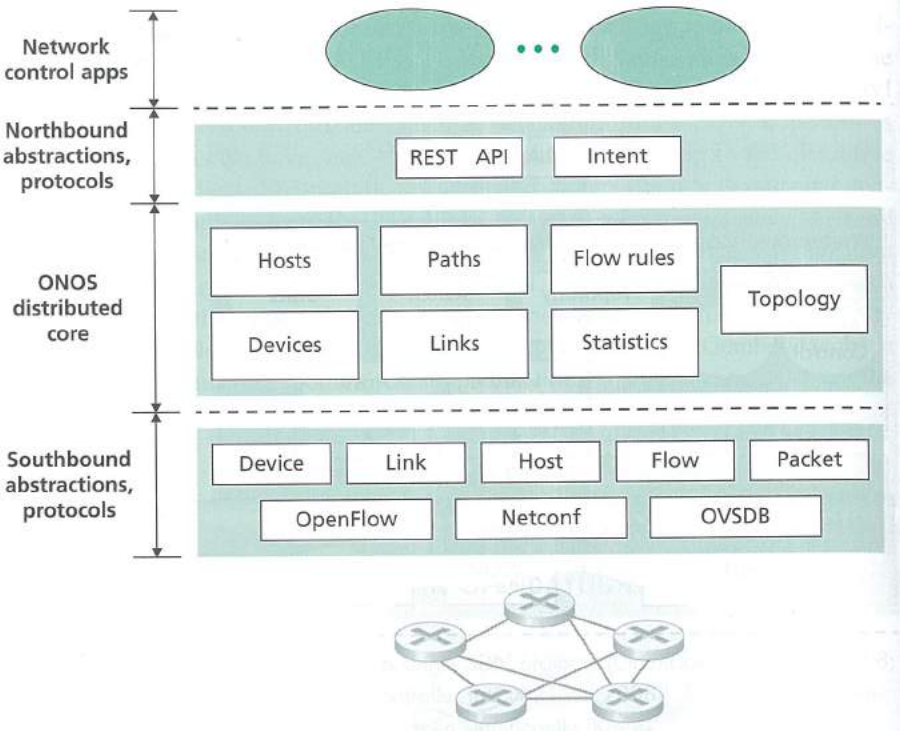


Figure 5.18 ♦ ONOS controller architecture

The ONOS Controller

Figure 5.18 presents a simplified view of the ONOS controller ONOS 2016]. Similar to the canonical controller in Figure 5.15, three layers can be identified in the ONOS controller:

- *Northbound abstractions and protocols.* A unique feature of ONOS is its intent framework, which allows an application to request a high-level service (e.g., to setup a connection between host A and Host B, or conversely to not allow Host A and host B to communicate) without having to know the details of how this service is performed. State information is provided to network-control applications across the northbound API either synchronously (via query) or asynchronously (via listener callbacks, e.g., when network state changes).
- *Distributed core.* The state of the network's links, hosts, and devices is maintained in ONOS's distributed core. ONOS is deployed as a service on a set of interconnected servers, with each server running an identical copy of the ONOS software; an increased number of servers offers an increased service capacity. The ONOS core

provides the mechanisms for service replication and coordination among instances, providing the applications above and the network devices below with the abstraction of logically centralized core services.

- *Southbound abstractions and protocols.* The southbound abstractions mask the heterogeneity of the underlying hosts, links, switches, and protocols, allowing the distributed core to be both device and protocol agnostic. Because of this abstraction, the southbound interface below the distributed core is logically higher than in our canonical controller in Figure 5.14 or the ODL controller in Figure 5.17.

5.6 ICMP: The Internet Control Message Protocol

The Internet Control Message Protocol (ICMP), specified in [RFC 792], is used by hosts and routers to communicate network-layer information to each other. The most typical use of ICMP is for error reporting. For example, when running an HTTP session, you may have encountered an error message such as "Destination network unreachable." This message had its origins in ICMP. At some point, an IP router was unable to find a path to the host specified in your HTTP request. That router created and sent an ICMP message to your host indicating the error.

ICMP is often considered part of IP, but architecturally it lies just above IP, as ICMP messages are carried inside IP datagrams. That is, ICMP messages are carried as IP payload, just as TCP or UDP segments are carried as IP payload. Similarly, when a host receives an IP datagram with ICMP specified as the upper-layer protocol (an upper-layer protocol number of 1), it demultiplexes the datagram's contents to ICMP, just as it would demultiplex a datagram's content to TCP or UDP.

ICMP messages have a type and a code field, and contain the header and the first 8 bytes of the IP datagram that caused the ICMP message to be generated in the first place (so that the sender can determine the datagram that caused the error). Selected ICMP message types are shown in Figure 5.19. Note that ICMP messages are used not only for signaling error conditions.

The well-known ping program sends an ICMP type 8 code 0 message to the specified host. The destination host, seeing the echo request, sends back a type 0 code 0 ICMP echo reply. Most TCP/IP implementations support the ping server directly in the operating system; that is, the server is not a process. Chapter 11 of [Stevens 1990] provides the source code for the ping client program. Note that the client program needs to be able to instruct the operating system to generate an ICMP message of type 8 code 0.

Another interesting ICMP message is the source quench message. This message is seldom used in practice. Its original purpose was to perform congestion control—to allow a congested router to send an ICMP source quench message to a host to force

ICMP Type	Code	Description
0	0	echo reply (to ping)
3	0	destination network unreachable
3	1	destination host unreachable
3	2	destination protocol unreachable
3	3	destination port unreachable
3	6	destination network unknown
3	7	destination host unknown
4	0	source quench (congestion control)
8	0	echo request
9	0	router advertisement
10	0	router discovery
11	0	TTL expired
12	0	IP header bad

Figure 5.19 ♦ ICMP message types

that host to reduce its transmission rate. We have seen in Chapter 3 that TCP has its own congestion-control mechanism that operates at the transport layer, without the use of network-layer feedback such as the ICMP source quench message.

In Chapter 1 we introduced the Traceroute program, which allows us to trace a route from a host to any other host in the world. Interestingly, Traceroute is implemented with ICMP messages. To determine the names and addresses of the routers between source and destination, Traceroute in the source sends a series of ordinary IP datagrams to the destination. Each of these datagrams carries a UDP segment with an unlikely UDP port number. The first of these datagrams has a TTL of 1, the second of 2, the third of 3, and so on. The source also starts timers for each of the datagrams. When the *n*th datagram arrives at the *n*th router, the *n*th router observes that the TTL of the datagram has just expired. According to the rules of the IP protocol, the router discards the datagram and sends an ICMP warning message to the source (type 11 code 0). This warning message includes the name of the router and its IP address. When this ICMP message arrives back at the source, the source obtains the round-trip time from the timer and the name and IP address of the *n*th router from the ICMP message.

How does a Traceroute source know when to stop sending UDP segments? Recall that the source increments the TTL field for each datagram it sends. Thus, one of the datagrams will eventually make it all the way to the destination host. Because this datagram contains a UDP segment with an unlikely port number, the destination

host sends a port unreachable ICMP message (type 3 code 3) back to the source. When the source host receives this particular ICMP message, it knows it does not need to send additional probe packets. (The standard Traceroute program actually sends sets of three packets with the same TTL; thus the Traceroute output provides three results for each TTL.)

In this manner, the source host learns the number and the identities of routers that lie between it and the destination host and the round-trip time between the two hosts. Note that the Traceroute client program must be able to instruct the operating system to generate UDP datagrams with specific TTL values and must also be able to be notified by its operating system when ICMP messages arrive. Now that you understand how Traceroute works, you may want to go back and play with it some more.

A new version of ICMP has been defined for IPv6 in RFC 4443. In addition to reorganizing the existing ICMP type and code definitions, ICMPv6 also added new types and codes required by the new IPv6 functionality. These include the "Packet Too Big" type and an "unrecognized IPv6 options" error code.

5.7 Network Management and SNMP

Having now made our way to the end of our study of the network layer, with only the link-layer before us, we're well aware that a network consists of many complex, interacting pieces of hardware and software—from the links, switches, routers, hosts, and other devices that comprise the physical components of the network to the many protocols that control and coordinate these devices. When hundreds or thousands of such components are brought together by an organization to form a network, the job of the network administrator to keep the network "up and running" is surely a challenge. We saw in Section 5.5 that the logically centralized controller can help with this process in an SDN context. But the challenge of network management has been around long before SDN, with a rich set of network management tools and approaches that help the network administrator monitor, manage, and control the network. We'll study these tools and techniques in this section.

An often-asked question is "What is network management?" A well-conceived, single-sentence (albeit a rather long run-on sentence) definition of network management from [Saydam 1996] is:

> *Network management includes the deployment, integration, and coordination of the hardware, software, and human elements to monitor, test, poll, configure, analyze, evaluate, and control the network and element resources to meet the real-time, operational performance, and Quality of Service requirements at a reasonable cost.*

Given this broad definition, we'll cover only the rudiments of network management in this section—the architecture, protocols, and information base used by

a network administrator in performing their task. We'll not cover the administrator's decision-making processes, where topics such as fault identification [Labovitz 1997; Steinder 2002; Feamster 2005; Wu 2005; Teixeira 2006], anomaly detection [Lakhina 2005; Barford 2009], network design/engineering to meet contracted Service Level Agreements (SLA's) [Huston 1999a], and more come into consideration. Our focus is thus purposefully narrow; the interested reader should consult these references, the excellent network-management text by Subramanian [Subramanian 2000], and the more detailed treatment of network management available on the Web site for this text.

5.7.1 The Network Management Framework

Figure 5.20 shows the key components of network management:

- The **managing server** is an application, typically with a human in the loop, running in a centralized network management station in the network operations center (NOC). The managing server is the locus of activity for network management; it controls the collection, processing, analysis, and/or display of network management information. It is here that actions are initiated to control network behavior and here that the human network administrator interacts with the network's devices.
- A **managed device** is a piece of network equipment (including its software) that resides on a managed network. A managed device might be a host, router, switch, middlebox, modem, thermometer, or other network-connected device. There may be several so-called **managed objects** within a managed device. These managed objects are the actual pieces of hardware within the managed device (for example, a network interface card is but one component of a host or router), and configuration parameters for these hardware and software components (for example, an intra-AS routing protocol such as OSPF).
- Each managed object within a managed device associated information that is collected into a **Management Information Base (MIB)**; we'll see that the values of these pieces of information are available to (and in many cases able to be set by) the managing server. A MIB object might be a counter, such as the number of IP datagrams discarded at a router due to errors in an IP datagram header, or the number of UDP segments received at a host; descriptive information such as the version of the software running on a DNS server; status information such as whether a particular device is functioning correctly; or protocol-specific information such as a routing path to a destination. MIB objects are specified in a data description language known as SMI (Structure of Management Information) [RFC 2578; RFC 2579; RFC 2580]. A formal definition language is used to ensure that the syntax and semantics of the network management data are well defined and unambiguous. Related MIB objects are gathered into MIB modules. As of mid-2015, there were nearly 400 MIB modules defined by RFCs, and a much larger number of vendor-specific (private) MIB modules.
- Also resident in each managed device is a **network management agent**, a process running in the managed device that communicates with the managing server,

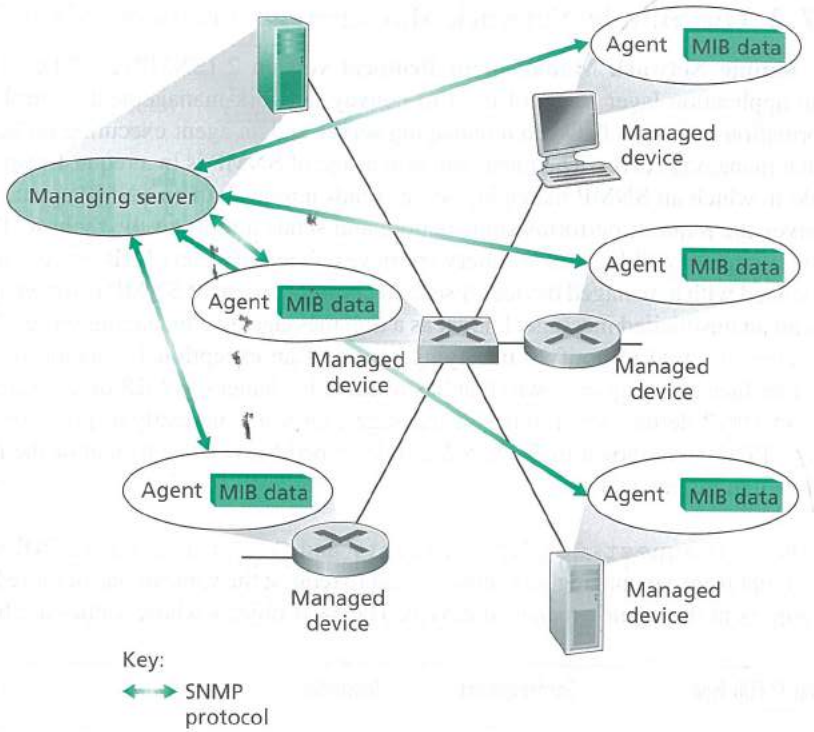


Figure 5.20 ♦ Elements of network management: Managing server, managed devices, MIB data, remote agents, SNMP

taking local actions at the managed device under the command and control of the managing server. The network management agent is similar to the routing agent that we saw in Figure 5.2.

- The final component of a network management framework is the **network management protocol**. The protocol runs between the managing server and the managed devices, allowing the managing server to query the status of managed devices and indirectly take actions at these devices via its agents. Agents can use the network management protocol to inform the managing server of exceptional events (for example, component failures or violation of performance thresholds). It's important to note that the network management protocol does not itself manage the network. Instead, it provides capabilities that a network administrator can use to manage ("monitor, test, poll, configure, analyze, evaluate, and control") the network. This is a subtle, but important, distinction. In the following section, we'll cover the Internet's SNMP (Simple Network Management Protocol) protocol.

5.7.2 The Simple Network Management Protocol (SNMP)

The **Simple Network Management Protocol** version 2 (SNMPv2) [RFC 3416] is an application-layer protocol used to convey network-management control and information messages between a managing server and an agent executing on behalf of that managing server. The most common usage of SNMP is in a request-response mode in which an SNMP managing server sends a request to an SNMP agent, who receives the request, performs some action, and sends a reply to the request. Typically, a request will be used to query (retrieve) or modify (set) MIB object values associated with a managed device. A second common usage of SNMP is for an agent to send an unsolicited message, known as a trap message, to a managing server. Trap messages are used to notify a managing server of an exceptional situation (e.g., a link interface going up or down) that has resulted in changes to MIB object values.

SNMPv2 defines seven types of messages, known generically as protocol data units—PDUs—as shown in Table 5.2 and described below. The format of the PDU is shown in Figure 5.21.

- The `GetRequest`, `GetNextRequest`, and `GetBulkRequest` PDUs are all sent from a managing server to an agent to request the value of one or more MIB objects at the agent's managed device. The MIB objects whose values are being

SNMPv2 PDU Type	Sender-receiver	Description
`GetRequest`	manager-to-agent	get value of one or more MIB object instances
`GetNextRequest`	manager-to-agent	get value of next MIB object instance in list or table
`GetBulkRequest`	manager-to-agent	get values in large block of data, for example, values in a large table
`InformRequest`	manager-to-manager	inform remote managing entity of MIB values remote to its access
`SetRequest`	manager-to-agent	set value of one or more MIB object instances
`Response`	agent-to-manager or manager-to-manager	generated in response to `GetRequest`, `GetNextRequest`, `GetBulkRequest`, `SetRequest` PDU, or `InformRequest`
`SNMPv2-Trap`	agent-to-manager	inform manager of an exceptional event #

Table 5.2 ♦ SNMPv2 PDU types

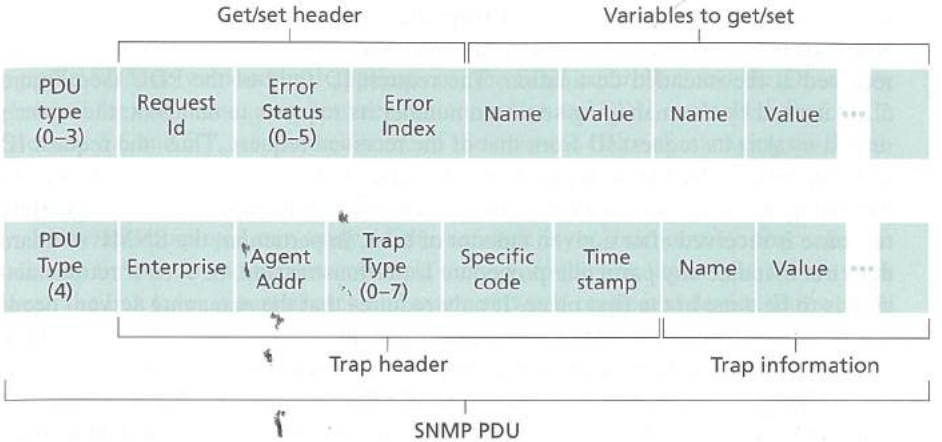


Figure 5.21 ♦ SNMP PDU format

requested are specified in the variable binding portion of the PDU. `GetRequest`, `GetNextRequest`, and `GetBulkRequest` differ in the granularity of their data requests. `GetRequest` can request an arbitrary set of MIB values; multiple `GetNextRequests` can be used to sequence through a list or table of MIB objects; `GetBulkRequest` allows a large block of data to be returned, avoiding the overhead incurred if multiple `GetRequest` or `GetNextRequest` messages were to be sent. In all three cases, the agent responds with a `Response PDU` containing the object identifiers and their associated values.

- The `SetRequest` PDU is used by a managing server to set the value of one or more MIB objects in a managed device. An agent replies with a `Response` PDU with the "noError" error status to confirm that the value has indeed been set.
- The `InformRequest` PDU is used by a managing server to notify another managing server of MIB information that is remote to the receiving server.
- The `Response PDU` is typically sent from a managed device to the managing server in response to a request message from that server, returning the requested information.
- The final type of SNMPv2 PDU is the trap message. Trap messages are generated asynchronously; that is, they are not generated in response to a received request but rather in response to an event for which the managing server requires notification. RFC 3418 defines well-known trap types that include a cold or warm start by a device, a link going up or down, the loss of a neighbor, or an authentication failure event. A received trap request has no required response from a managing server.

Given the request-response nature of SNMP, it is worth noting here that although SNMP PDUs can be carried via many different transport protocols, the SNMP PDU is typically carried in the payload of a UDP datagram. Indeed, RFC 3417 states

that UDP is "the preferred transport mapping." However, since UDP is an unreliable transport protocol, there is no guarantee that a request, or its response, will be received at the intended destination. The request ID field of the PDU (see Figure 5.21) is used by the managing server to number its requests to an agent; the agent's response takes its request ID from that of the received request. Thus, the request ID field can be used by the managing server to detect lost requests or replies. It is up to the managing server to decide whether to retransmit a request if no corresponding response is received after a given amount of time. In particular, the SNMP standard does not mandate any particular procedure for retransmission, or even if retransmission is to be done in the first place. It only requires that the managing server "needs to act responsibly in respect to the frequency and duration of retransmissions." This, of course, leads one to wonder how a "responsible" protocol should act!

SNMP has evolved through three versions. The designers of SNMPv3 have said that "SNMPv3 can be thought of as SNMPv2 with additional security and administration capabilities" [RFC 3410]. Certainly, there are changes in SNMPv3 over SNMPv2, but nowhere are those changes more evident than in the area of administration and security. The central role of security in SNMPv3 was particularly important, since the lack of adequate security resulted in SNMP being used primarily for monitoring rather than control (for example, `SetRequest` is rarely used in SNMPv1). Once again, we see that security—a topic we'll cover in detail in Chapter 8 — is of critical concern, but once again a concern whose importance had been realized perhaps a bit late and only then "added on."

5.7 Summary

We have now completed our two-chapter journey into the network core—a journey that began with our study of the network layer's data plane in Chapter 4 and finished here with our study of the network layer's control plane. We learned that the control plane is the network-wide logic that controls not only how a datagram is forwarded among routers along an end-to-end path from the source host to the destination host, but also how network-layer components and services are configured and managed.

We learned that there are two broad approaches towards building a control plane: traditional *per-router control* (where a routing algorithm runs in each and every router and the routing component in the router communicates with the routing components in other routers) and *software-defined networking* (SDN) control (where a logically centralized controller computes and distributes the forwarding tables to be used by each and every router). We studied two fundamental routing algorithms for computing least cost paths in a graph—link-state routing and distance-vector routing—in Section 5.2; these algorithms find application in both per-router control and in SDN control. These algorithms are the basis for two widely-deployed Internet

routing protocols, OSPF and BGP, that we covered in Sections 5.3 and 5.4. We covered the SDN approach to the network-layer control plane in Section 5.5, investigating SDN network-control applications, the SDN controller, and the OpenFlow protocol for communicating between the controller and SDN-controlled devices. In Sections 5.6 and 5.7, we covered some of the nuts and bolts of managing an IP network: ICMP (the Internet Control Message Protocol) and SNMP (the Simple Network Management Protocol).

Having completed our study of the network layer, our journey now takes us one step further down the protocol stack, namely, to the link layer. Like the network layer, the link layer is part of each and every network-connected device. But we will see in the next chapter that the link layer has the much more localized task of moving packets between nodes on the same link or LAN. Although this task may appear on the surface to be rather simple compared with that of the network layer's tasks, we will see that the link layer involves a number of important and fascinating issues that can keep us busy for a long time.

Homework Problems and Questions

Chapter 5 Review Questions

SECTION 5.1

R1. What is meant by a control plane that is based on per-router control? In such cases, when we say the network control and data planes are implemented "monolithically," what do we mean?

R2. What is meant by a control plane that is based on logically centralized control? In such cases, are the data plane and the control plane implemented within the same device or in separate devices? Explain.

SECTION 5.2

R3. Compare and contrast the properties of a centralized and a distributed routing algorithm. Give an example of a routing protocol that takes a centralized and a decentralized approach.

R4. Compare and contrast link-state and distance-vector routing algorithms.

R5. What is the "count to infinity" problem in distance vector routing?

R6. Is it necessary that every autonomous system use the same intra-AS routing algorithm? Why or why not?

SECTIONS 5.3–5.4

R7. Why are different inter-AS and intra-AS protocols used in the Internet?

R8. True or false: When an OSPF route sends its link state information, it is sent only to those nodes directly attached neighbors. Explain.

R9. What is meant by an *area* in an OSPF autonomous system? Why was the concept of an area introduced?

R10. Define and contrast the following terms: *subnet, prefix*, and *BGP route*.

R11. How does BGP use the NEXT-HOP attribute? How does it use the AS-PATH attribute?

R12. Describe how a network administrator of an upper-tier ISP can implement policy when configuring BGP.

R13. True or false: When a BGP router receives an advertised path from its neighbor, it must add its own identity to the received path and then send that new path on to all of its neighbors. Explain.

SECTION 5.5

R14. Describe the main role of the communication layer, the network-wide state-management layer, and the network-control application layer in an SDN controller.

R15. Suppose you wanted to implement a new routing protocol in the SDN control plane. At which layer would you implement that protocol? Explain.

R16. What types of messages flow across an SDN controller's northbound and southbound APIs? Who is the recipient of these messages sent from the controller across the southbound interface, and who sends messages to the controller across the northbound interface?

R17. Describe the purpose of two types of OpenFlow messages (of your choosing) that are sent from a controlled device to the controller. Describe the purpose of two types of Openflow messages (of your choosing) that are send from the controller to a controlled device.

R18. What is the purpose of the service abstraction layer in the OpenDaylight SDN controller?

SECTIONS 5.6–5.7

R19. Names four different types of ICMP messages

R20. What two types of ICMP messages are received at the sending host executing the *Traceroute* program?

R21. Define the following terms in the context of SNMP: *managing server, managed device, network management agent and MIB.*

R22. What are the purposes of the SNMP *GetRequest* and *SetRequest* messages?

R23. What is the purpose of the SNMP trap message?

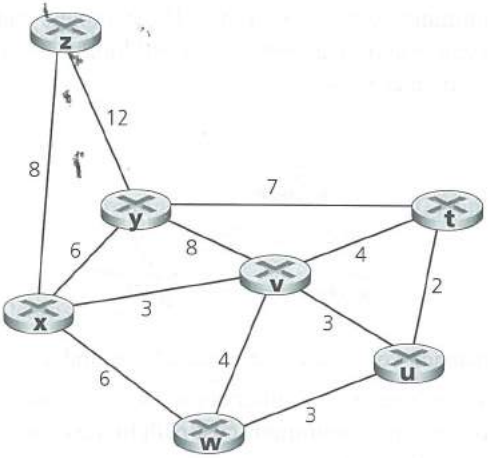
Problems

P1. Looking at Figure 5.3, enumerate the paths from y to u that do not contain any loops.

P2. Repeat Problem P1 for paths from x to z, z to u, and z to w.

P3. Consider the following network. With the indicated link costs, use Dijkstra's shortest-path algorithm to compute the shortest path from x to all network nodes. Show how the algorithm works by computing a table similar to Table 5.1.

VideoNote
Dijkstra's algorithm: discussion and example



P4. Consider the network shown in Problem P3. Using Dijkstra's algorithm, and showing your work using a table similar to Table 5.1, do the following:

a. Compute the shortest path from t to all network nodes.

b. Compute the shortest path from u to all network nodes.

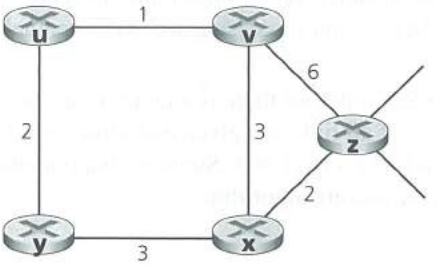
c. Compute the shortest path from v to all network nodes.

d. Compute the shortest path from w to all network nodes.

e. Compute the shortest path from y to all network nodes.

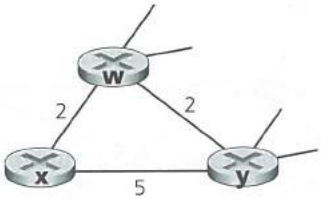
f. Compute the shortest path from z to all network nodes.

P5. Consider the network shown below, and assume that each node initially knows the costs to each of its neighbors. Consider the distance-vector algorithm and show the distance table entries at node z.



P6. Consider a general topology (that is, not the specific network shown above) and a synchronous version of the distance-vector algorithm. Suppose that at each iteration, a node exchanges its distance vectors with its neighbors and receives their distance vectors. Assuming that the algorithm begins with each node knowing only the costs to its immediate neighbors, what is the maximum number of iterations required before the distributed algorithm converges? Justify your answer.

P7. Consider the network fragment shown below. x has only two attached neighbors, w and y. w has a minimum-cost path to destination u (not shown) of 5, and y has a minimum-cost path to u of 6. The complete paths from w and y to u (and between w and y) are not shown. All link costs in the network have strictly positive integer values.



a. Give x's distance vector for destinations w, y, and u.

b. Give a link-cost change for either $c(x,w)$ or $c(x,y)$ such that x will inform its neighbors of a new minimum-cost path to u as a result of executing the distance-vector algorithm.

c. Give a link-cost change for either $c(x,w)$ or $c(x,y)$ such that x will *not* inform its neighbors of a new minimum-cost path to u as a result of executing the distance-vector algorithm.

P8. Consider the three-node topology shown in Figure 5.6. Rather than having the link costs shown in Figure 5.6, the link costs are $c(x,y) = 3$, $c(y,z) = 6$, $c(z,x) = 4$. Compute the distance tables after the initialization step and after each iteration of a synchronous version of the distance-vector algorithm (as we did in our earlier discussion of Figure 5.6).

P9. Consider the count-to-infinity problem in the distance vector routing. Will the count-to-infinity problem occur if we decrease the cost of a link? Why? How about if we connect two nodes which do not have a link?

P10. Argue that for the distance-vector algorithm in Figure 5.6, each value in the distance vector $D(x)$ is non-increasing and will eventually stabilize in a finite number of steps.

P11. Consider Figure 5.7. Suppose there is another router w, connected to router y and z. The costs of all links are given as follows: $c(x,y) = 4$, $c(x,z) = 50$, $c(y,w) = 1$, $c(z,w) = 1$, $c(y,z) = 3$. Suppose that poisoned reverse is used in the distance-vector routing algorithm.

a. When the distance vector routing is stabilized, router w, y, and z inform their distances to x to each other. What distance values do they tell each other?

b. Now suppose that the link cost between x and y increases to 60. Will there be a count-to-infinity problem even if poisoned reverse is used? Why or why not? If there is a count-to-infinity problem, then how many iterations are needed for the distance-vector routing to reach a stable state again? Justify your answer.

c. How do you modify $c(y,z)$ such that there is no count-to-infinity problem at all if $c(y,x)$ changes from 4 to 60?

P12. Describe how loops in paths can be detected in BGP.

P13. Will a BGP router always choose the loop-free route with the shortest ASpath length? Justify your answer.

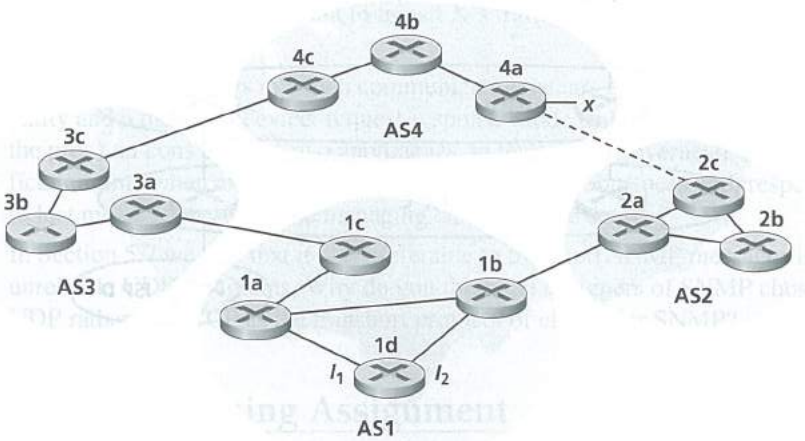
P14. Consider the network shown below. Suppose AS3 and AS2 are running OSPF for their intra-AS routing protocol. Suppose AS1 and AS4 are running RIP for their intra-AS routing protocol. Suppose eBGP and iBGP are used for the inter-AS routing protocol. Initially suppose there is *no* physical link between AS2 and AS4.

a. Router 3c learns about prefix x from which routing protocol: OSPF, RIP, eBGP, or iBGP?

b. Router 3a learns about x from which routing protocol?

c. Router 1c learns about x from which routing protocol?

d. Router 1d learns about x from which routing protocol?



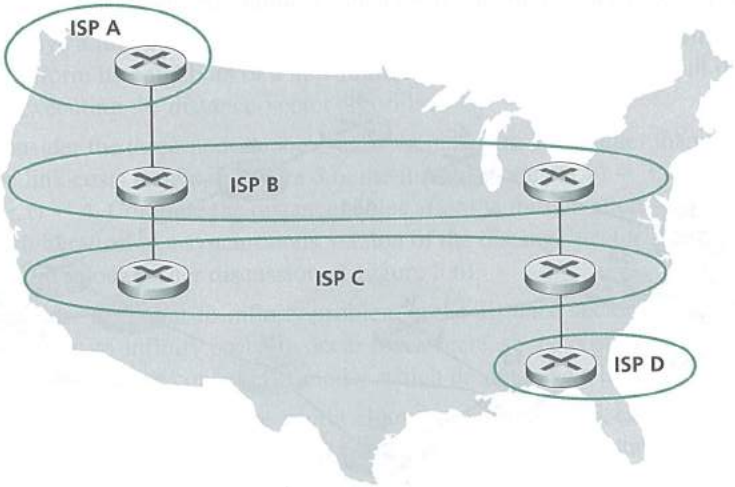
P15. Referring to the previous problem, once router 1d learns about x it will put an entry (x, I) in its forwarding table.

a. Will I be equal to I_1 or I_2 for this entry? Explain why in one sentence.

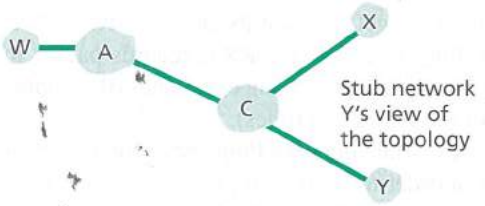
b. Now suppose that there is a physical link between AS2 and AS4, shown by the dotted line. Suppose router 1d learns that x is accessible via AS2 as well as via AS3. Will I be set to I_1 or I_2? Explain why in one sentence.

c. Now suppose there is another AS, called AS5, which lies on the path between AS2 and AS4 (not shown in diagram). Suppose router 1d learns that x is accessible via AS2 AS5 AS4 as well as via AS3 AS4. Will I be set to I_1 or I_2? Explain why in one sentence.

P16. Consider the following network. ISP B provides national backbone service to regional ISP A. ISP C provides national backbone service to regional ISP D. Each ISP consists of one AS. B and C peer with each other in two places using BGP. Consider traffic going from A to D. B would prefer to hand that traffic over to C on the West Coast (so that C would have to absorb the cost of carrying the traffic cross-country), while C would prefer to get the traffic via its East Coast peering point with B (so that B would have carried the traffic across the country). What BGP mechanism might C use, so that B would hand over A-to-D traffic at its East Coast peering point? To answer this question, you will need to dig into the BGP specification.



P17. In Figure 5.13, consider the path information that reaches stub networks W, X, and Y. Based on the information available at W and X, what are their respective views of the network topology? Justify your answer. The topology view at Y is shown below.



P18. Consider Figure 5.13. B would never forward traffic destined to Y via X based on BGP routing. But there are some very popular applications for which data packets go to X first and then flow to Y. Identify one such application, and describe how data packets follow a path not given by BGP routing.

P19. In Figure 5.13, suppose that there is another stub network V that is a customer of ISP A. Suppose that B and C have a peering relationship, and A is a customer of both B and C. Suppose that A would like to have the traffic destined to W to come from B only, and the traffic destined to V from either B or C. How should A advertise its routes to B and C? What AS routes does C receive?

P20. Suppose ASs X and Z are not directly connected but instead are connected by AS Y. Further suppose that X has a peering agreement with Y, and that Y has a peering agreement with Z. Finally, suppose that Z wants to transit all of Y's traffic but does not want to transit X's traffic. Does BGP allow Z to implement this policy?

P21. Consider the two ways in which communication occurs between a managing entity and a managed device: request-response mode and trapping. What are the pros and cons of these two approaches, in terms of (1) overhead, (2) notification time when exceptional events occur, and (3) robustness with respect to lost messages between the managing entity and the device?

P22. In Section 5.7 we saw that it was preferable to transport SNMP messages in unreliable UDP datagrams. Why do you think the designers of SNMP chose UDP rather than TCP as the transport protocol of choice for SNMP?

Socket Programming Assignment

At the end of Chapter 2, there are four socket programming assignments. Below, you will find a fifth assignment which employs ICMP, a protocol discussed in this chapter.

Assignment 5: ICMP Ping

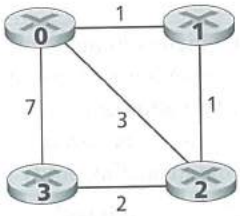
Ping is a popular networking application used to test from a remote location whether a particular host is up and reachable. It is also often used to measure latency between the client host and the target host. It works by sending ICMP "echo request" packets (i.e., ping packets) to the target host and listening for ICMP "echo response" replies (i.e., pong packets). Ping measures the RRT, records packet loss, and calculates a statistical summary of multiple ping-pong exchanges (the minimum, mean, max, and standard deviation of the round-trip times).

In this lab, you will write your own Ping application in Python. Your application will use ICMP. But in order to keep your program simple, you will not exactly follow the official specification in RFC 1739. Note that you will only need to write the client side of the program, as the functionality needed on the server side is built into almost all operating systems. You can find full details of this assignment, as well as important snippets of the Python code, at the Web site http://www.pearsonhighered.com/cs-resources.

Programming Assignment

In this programming assignment, you will be writing a "distributed" set of procedures that implements a distributed asynchronous distance-vector routing for the network shown below.

You are to write the following routines that will "execute" asynchronously within the emulated environment provided for this assignment. For node 0, you will write the routines:



- *rtinit0()*. This routine will be called once at the beginning of the emulation. *rtinit0()* has no arguments. It should initialize your distance table in node 0 to reflect the direct costs of 1, 3, and 7 to nodes 1, 2, and 3, respectively. In the figure above, all links are bidirectional and the costs in both directions are identical. After initializing the distance table and any other data structures needed by your node 0 routines, it should then send its directly connected neighbors (in this case, 1, 2, and 3) the cost of its minimum-cost paths to all other network nodes.

This minimum-cost information is sent to neighboring nodes in a routing update packet by calling the routine *tolayer2()*, as described in the full assignment. The format of the routing update packet is also described in the full assignment.

- *rtupdate0(struct rtpkt *rcvdpkt)*. This routine will be called when node 0 receives a routing packet that was sent to it by one of its directly connected neighbors. The parameter **rcvdpkt* is a pointer to the packet that was received. *rtupdate0()* is the "heart" of the distance-vector algorithm. The values it receives in a routing update packet from some other node *i* contain *i*'s current shortest-path costs to all other network nodes. *rtupdate0()* uses these received values to update its own distance table (as specified by the distance-vector algorithm). If its own minimum cost to another node changes as a result of the update, node 0 informs its directly connected neighbors of this change in minimum cost by sending them a routing packet. Recall that in the distance-vector algorithm, only directly connected nodes will exchange routing packets. Thus, nodes 1 and 2 will communicate with each other, but nodes 1 and 3 will not communicate with each other.

Similar routines are defined for nodes 1, 2, and 3. Thus, you will write eight procedures in all: *rtinit0(), rtinit1(), rtinit2(), rtinit3(), rtupdate0(), rtupdate1(), rtupdate2()*, and *rtupdate3()*. These routines will together implement a distributed, asynchronous computation of the distance tables for the topology and costs shown in the figure on the preceding page.

You can find the full details of the programming assignment, as well as C code that you will need to create the simulated hardware/software environment, at http://www.pearsonhighered.com/cs-resource. A Java version of the assignment is also available.

Wireshark Lab

In the Web site for this textbook, www.pearsonhighered.com/cs-resources, you'll find a Wireshark lab assignment that examines the use of the ICMP protocol in the ping and traceroute commands.

AN INTERVIEW WITH...

Jennifer Rexford

Jennifer Rexford is a Professor in the Computer Science department at Princeton University. Her research has the broad goal of making computer networks easier to design and manage, with particular emphasis on routing protocols. From 1996–2004, she was a member of the Network Management and Performance department at AT&T Labs–Research. While at AT&T, she designed techniques and tools for network measurement, traffic engineering, and router configuration that were deployed in AT&T's backbone network. Jennifer is co-author of the book "Web Protocols and Practice: Networking Protocols, Caching, and Traffic Measurement," published by Addison-Wesley in May 2001. She served as the chair of ACM SIGCOMM from 2003 to 2007. She received her BSE degree in electrical engineering from Princeton University in 1991, and her PhD degree in electrical engineering and computer science from the University of Michigan in 1996. In 2004, Jennifer was the winner of ACM's Grace Murray Hopper Award for outstanding young computer professional and appeared on the MIT TR-100 list of top innovators under the age of 35.

Please describe one or two of the most exciting projects you have worked on during your career. What were the biggest challenges?

When I was a researcher at AT&T, a group of us designed a new way to manage routing in Internet Service Provider backbone networks. Traditionally, network operators configure each router individually, and these routers run distributed protocols to compute paths through the network. We believed that network management would be simpler and more flexible if network operators could exercise direct control over how routers forward traffic based on a network-wide view of the topology and traffic. The Routing Control Platform (RCP) we designed and built could compute the routes for all of AT&T's backbone on a single commodity computer, and could control legacy routers without modification. To me, this project was exciting because we had a provocative idea, a working system, and ultimately a real deployment in an operational network. Fast forward a few years, and software-defined networking (SDN) has become a mainstream technology, and standard protocols (like OpenFlow) have made it much easier to tell the underlying switches what to do.

How do you think software-defined networking should evolve in the future?

In a major break from the past, control-plane software can be created by many different programmers, not just at companies selling network equipment. Yet, unlike the applications running on a server or a smart phone, controller apps must work *together* to handle the same traffic. Network operators do not want to perform load balancing on some traffic and routing on other traffic; instead, they want to perform load balancing and routing, together, on the same traffic. Future SDN controller platforms should offer good programming abstractions for *composing* independently written multiple controller applications together. More broadly, good programming abstractions can make it easier to create controller applications, without having to worry about low-level details like flow table entries, traffic counters, bit patterns in packet headers, and so on. Also, while an SDN controller is logically centralized, the network still consists of a distributed collection of devices. Future controllers should offer good abstractions for updating the flow tables across the network, so apps can reason about what happens to packets in flight while the devices are updated. Programming abstractions for control-plane software is an exciting area for interdisciplinary research between computer networking, distributed systems, and programming languages, with a real chance for practical impact in the years ahead.

Where do you see the future of networking and the Internet?

Networking is an exciting field because the applications and the underlying technologies change all the time. We are always reinventing ourselves! Who would have predicted even ten years ago the dominance of smart phones, allowing mobile users to access existing applications as well as new location-based services? The emergence of cloud computing is fundamentally changing the relationship between users and the applications they run, and networked sensors and actuators (the "Internet of Things") are enabling a wealth of new applications (and security vulnerabilities!). The pace of innovation is truly inspiring.

The underlying network is a crucial component in all of these innovations. Yet, the network is notoriously "in the way"—limiting performance, compromising reliability, constraining applications, and complicating the deployment and management of services. We should strive to make the network of the future as invisible as the air we breathe, so it never stands in the way of new ideas and valuable services. To do this, we need to raise the level of abstraction above individual network devices and protocols (and their attendant acronyms!), so we can reason about the network and the user's high-level goals as a whole.

What people inspired you professionally?

I've long been inspired by Sally Floyd at the International Computer Science Institute. Her research is always purposeful, focusing on the important challenges facing the Internet. She digs deeply into hard questions until she understands the problem and the space of solutions

completely, and she devotes serious energy into "making things happen," such as pushing her ideas into protocol standards and network equipment. Also, she gives back to the community, through professional service in numerous standards and research organizations and by creating tools (such as the widely used ns-2 and ns-3 simulators) that enable other researchers to succeed. She retired in 2009 but her influence on the field will be felt for years to come.

What are your recommendations for students who want careers in computer science and networking?

Networking is an inherently interdisciplinary field. Applying techniques from other disciplines breakthroughs in networking come from such diverse areas as queuing theory, game theory, control theory, distributed systems, network optimization, programming languages, machine learning, algorithms, data structures, and so on. I think that becoming conversant in a related field, or collaborating closely with experts in those fields, is a wonderful way to put networking on a stronger foundation, so we can learn how to build networks that are worthy of society's trust. Beyond the theoretical disciplines, networking is exciting because we create real artifacts that real people use. Mastering how to design and build systems—by gaining experience in operating systems, computer architecture, and so on—is another fantastic way to amplify your knowledge of networking to help make the world a better place.

CHAPTER 6

The Link Layer and LANs

In the previous two chapters we learned that the network layer provides a communication service between *any* two network hosts. Between the two hosts, datagrams travel over a series of communication links, some wired and some wireless, starting at the source host, passing through a series of packet switches (switches and routers) and ending at the destination host. As we continue down the protocol stack, from the network layer to the link layer, we naturally wonder how packets are sent across the *individual links* that make up the end-to-end communication path. How are the network-layer datagrams encapsulated in the link-layer frames for transmission over a single link? Are different link-layer protocols used in the different links along the communication path? How are transmission conflicts in broadcast links resolved? Is there addressing at the link layer and, if so, how does the link-layer addressing operate with the network-layer addressing we learned about in Chapter 4? And what exactly is the difference between a switch and a router? We'll answer these and other important questions in this chapter.

In discussing the link layer, we'll see that there are two fundamentally different types of link-layer channels. The first type are broadcast channels, which connect multiple hosts in wireless LANs, satellite networks, and hybrid fiber-coaxial cable (HFC) access networks. Since many hosts are connected to the same broadcast communication channel, a so-called medium access protocol is needed to coordinate frame transmission. In some cases, a central controller may be used to coordinate

transmissions; in other cases, the hosts themselves coordinate transmissions. The second type of link-layer channel is the point-to-point communication link, such as that often found between two routers connected by a long-distance link, or between a user's office computer and the nearby Ethernet switch to which it is connected. Coordinating access to a point-to-point link is simpler; the reference material on this book's Web site has a detailed discussion of the Point-to-Point Protocol (PPP), which is used in settings ranging from dial-up service over a telephone line to high-speed point-to-point frame transport over fiber-optic links.

We'll explore several important link-layer concepts and technologies in this chapter. We'll dive deeper into error detection and correction, a topic we touched on briefly in Chapter 3. We'll consider multiple access networks and switched LANs, including Ethernet—by far the most prevalent wired LAN technology. We'll also look at virtual LANs, and data center networks. Although WiFi, and more generally wireless LANs, are link-layer topics, we'll postpone our study of these important topics until Chapter 7.

6.1 Introduction to the Link Layer

Let's begin with some important terminology. We'll find it convenient in this chapter to refer to any device that runs a link-layer (i.e., layer 2) protocol as a **node**. Nodes include hosts, routers, switches, and WiFi access points (discussed in Chapter 7). We will also refer to the communication channels that connect adjacent nodes along the communication path as **links**. In order for a datagram to be transferred from source host to destination host, it must be moved over each of the *individual links* in the end-to-end path. As an example, in the company network shown at the bottom of Figure 6.1, consider sending a datagram from one of the wireless hosts to one of the servers. This datagram will actually pass through six links: a WiFi link between sending host and WiFi access point, an Ethernet link between the access point and a link-layer switch; a link between the link-layer switch and the router, a link between the two routers; an Ethernet link between the router and a link-layer switch; and finally an Ethernet link between the switch and the server. Over a given link, a transmitting node encapsulates the datagram in a **link-layer frame** and transmits the frame into the link.

In order to gain further insight into the link layer and how it relates to the network layer, let's consider a transportation analogy. Consider a travel agent who is planning a trip for a tourist traveling from Princeton, New Jersey, to Lausanne, Switzerland. The travel agent decides that it is most convenient for the tourist to take a limousine from Princeton to JFK airport, then a plane from JFK airport to Geneva's airport, and finally a train from Geneva's airport to Lausanne's train station. Once the travel agent makes the three reservations, it is the responsibility of the Princeton limousine company to get the tourist from Princeton to JFK; it is the responsibility of the airline company to get the tourist from JFK to Geneva; and it is the responsibility

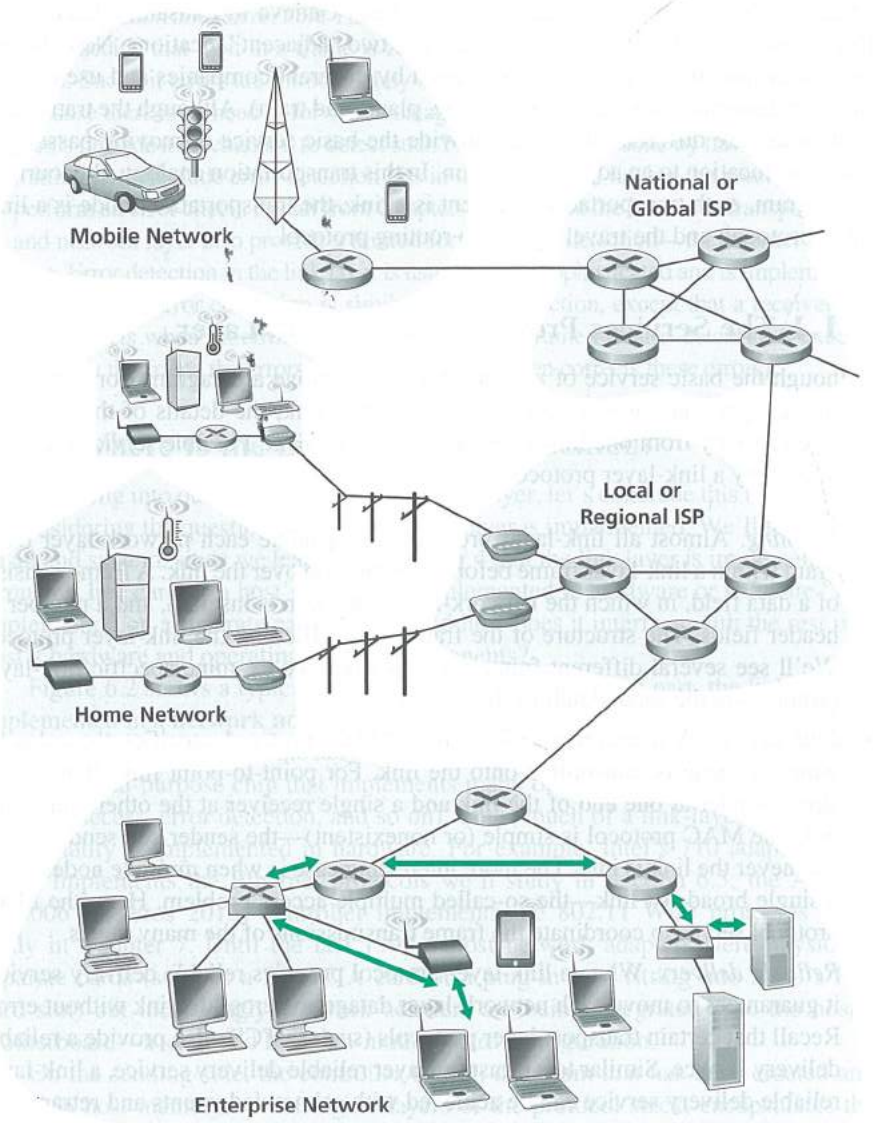


Figure 6.1 ♦ Six link-layer hops between wireless host and server

of the Swiss train service to get the tourist from Geneva to Lausanne. Each of the three segments of the trip is "direct" between two "adjacent" locations. Note that the three transportation segments are managed by different companies and use entirely different transportation modes (limousine, plane, and train). Although the transportation modes are different, they each provide the basic service of moving passengers from one location to an adjacent location. In this transportation analogy, the tourist is a datagram, each transportation segment is a link, the transportation mode is a link-layer protocol, and the travel agent is a routing protocol.

6.1.1 The Services Provided by the Link Layer

Although the basic service of any link layer is to move a datagram from one node to an adjacent node over a single communication link, the details of the provided service can vary from one link-layer protocol to the next. Possible services that can be offered by a link-layer protocol include:

- *Framing.* Almost all link-layer protocols encapsulate each network-layer datagram within a link-layer frame before transmission over the link. A frame consists of a data field, in which the network-layer datagram is inserted, and a number of header fields. The structure of the frame is specified by the link-layer protocol. We'll see several different frame formats when we examine specific link-layer protocols in the second half of this chapter.
- *Link access.* A medium access control (MAC) protocol specifies the rules by which a frame is transmitted onto the link. For point-to-point links that have a single sender at one end of the link and a single receiver at the other end of the link, the MAC protocol is simple (or nonexistent)—the sender can send a frame whenever the link is idle. The more interesting case is when multiple nodes share a single broadcast link—the so-called multiple access problem. Here, the MAC protocol serves to coordinate the frame transmissions of the many nodes.
- *Reliable delivery.* When a link-layer protocol provides reliable delivery service, it guarantees to move each network-layer datagram across the link without error. Recall that certain transport-layer protocols (such as TCP) also provide a reliable delivery service. Similar to a transport-layer reliable delivery service, a link-layer reliable delivery service can be achieved with acknowledgments and retransmissions (see Section 3.4). A link-layer reliable delivery service is often used for links that are prone to high error rates, such as a wireless link, with the goal of correcting an error locally—on the link where the error occurs—rather than forcing an end-to-end retransmission of the data by a transport- or application-layer protocol. However, link-layer reliable delivery can be considered an unnecessary overhead for low bit-error links, including fiber, coax, and many twisted-pair copper links. For this reason, many wired link-layer protocols do not provide a reliable delivery service.

- *Error detection and correction.* The link-layer hardware in a receiving node can incorrectly decide that a bit in a frame is zero when it was transmitted as a one, and vice versa. Such bit errors are introduced by signal attenuation and electromagnetic noise. Because there is no need to forward a datagram that has an error, many link-layer protocols provide a mechanism to detect such bit errors. This is done by having the transmitting node include error-detection bits in the frame, and having the receiving node perform an error check. Recall from Chapters 3 and 4 that the Internet's transport layer and network layer also provide a limited form of error detection—the Internet checksum. Error detection in the link layer is usually more sophisticated and is implemented in hardware. Error correction is similar to error detection, except that a receiver not only detects when bit errors have occurred in the frame but also determines exactly where in the frame the errors have occurred (and then corrects these errors).

6.1.2 Where Is the Link Layer Implemented?

Before diving into our detailed study of the link layer, let's conclude this introduction by considering the question of where the link layer is implemented. We'll focus here on an end system, since we learned in Chapter 4 that the link layer is implemented in a router's line card. Is a host's link layer implemented in hardware or software? Is it implemented on a separate card or chip, and how does it interface with the rest of a host's hardware and operating system components?

Figure 6.2 shows a typical host architecture. For the most part, the link layer is implemented in a **network adapter**, also sometimes known as a **network interface card (NIC)**. At the heart of the network adapter is the link-layer controller, usually a single, special-purpose chip that implements many of the link-layer services (framing, link access, error detection, and so on). Thus, much of a link-layer controller's functionality is implemented in hardware. For example, Intel's 710 adapter [Intel 2016] implements the Ethernet protocols we'll study in Section 6.5; the Atheros AR5006 [Atheros 2016] controller implements the 802.11 WiFi protocols we'll study in Chapter 7. Until the late 1990s, most network adapters were physically separate cards (such as a PCMCIA card or a plug-in card fitting into a PC's PCI card slot) but increasingly, network adapters are being integrated onto the host's motherboard—a so-called LAN-on-motherboard configuration.

On the sending side, the controller takes a datagram that has been created and stored in host memory by the higher layers of the protocol stack, encapsulates the datagram in a link-layer frame (filling in the frame's various fields), and then transmits the frame into the communication link, following the link-access protocol. On the receiving side, a controller receives the entire frame, and extracts the network-layer datagram. If the link layer performs error detection, then it is the sending controller that sets the error-detection bits in the frame header and it is the receiving controller that performs error detection.

Figure 6.2 shows a network adapter attaching to a host's bus (e.g., a PCI or PCI-X bus), where it looks much like any other I/O device to the other host

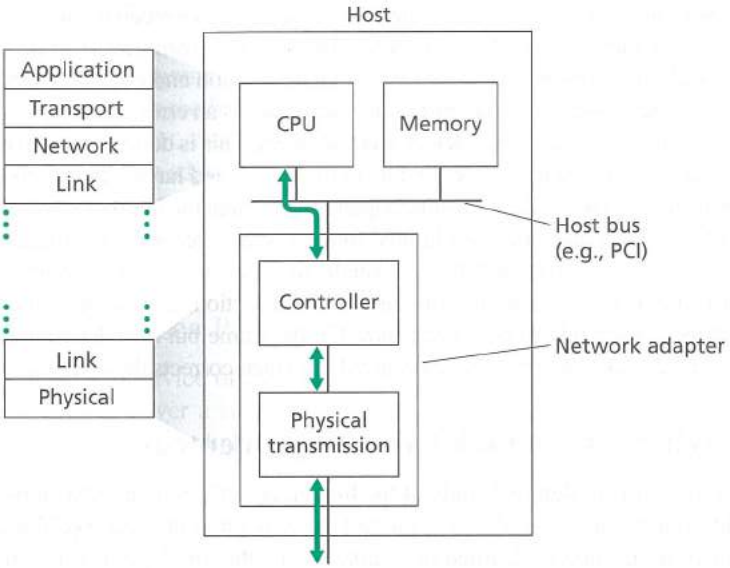


Figure 6.2 ♦ Network adapter: Its relationship to other host components and to protocol stack functionality

components. Figure 6.2 also shows that while most of the link layer is implemented in hardware, part of the link layer is implemented in software that runs on the host's CPU. The software components of the link layer implement higher-level link-layer functionality such as assembling link-layer addressing information and activating the controller hardware. On the receiving side, link-layer software responds to controller interrupts (e.g., due to the receipt of one or more frames), handling error conditions and passing a datagram up to the network layer. Thus, the link layer is a combination of hardware and software—the place in the protocol stack where software meets hardware. [Intel 2016] provides a readable overview (as well as a detailed description) of the XL710 controller from a software-programming point of view.

6.2 Error-Detection and -Correction Techniques

In the previous section, we noted that **bit-level error detection and correction**—detecting and correcting the corruption of bits in a link-layer frame sent from one node to another physically connected neighboring node—are two services often provided by the link layer. We saw in Chapter 3 that error-detection and -correction services are also often offered at the transport layer as well. In this section, we'll

examine a few of the simplest techniques that can be used to detect and, in some cases, correct such bit errors. A full treatment of the theory and implementation of this topic is itself the topic of many textbooks (for example, [Schwartz 1980] or [Bertsekas 1991]), and our treatment here is necessarily brief. Our goal here is to develop an intuitive feel for the capabilities that error-detection and -correction techniques provide and to see how a few simple techniques work and are used in practice in the link layer.

Figure 6.3 illustrates the setting for our study. At the sending node, data, D, to be protected against bit errors is augmented with error-detection and -correction bits (EDC). Typically, the data to be protected includes not only the datagram passed down from the network layer for transmission across the link, but also link-level addressing information, sequence numbers, and other fields in the link frame header. Both D and EDC are sent to the receiving node in a link-level frame. At the receiving node, a sequence of bits, D' and EDC' is received. Note that D' and EDC' may differ from the original D and EDC as a result of in-transit bit flips.

The receiver's challenge is to determine whether or not D' is the same as the original D, given that it has only received D' and EDC'. The exact wording of the receiver's decision in Figure 6.3 (we ask whether an error is detected, not whether an error has occurred!) is important. Error-detection and -correction techniques allow the receiver to sometimes, *but not always*, detect that bit errors have occurred. Even with the use of error-detection bits there still may be **undetected bit errors**; that is, the receiver may be unaware that the received information contains bit errors. As a

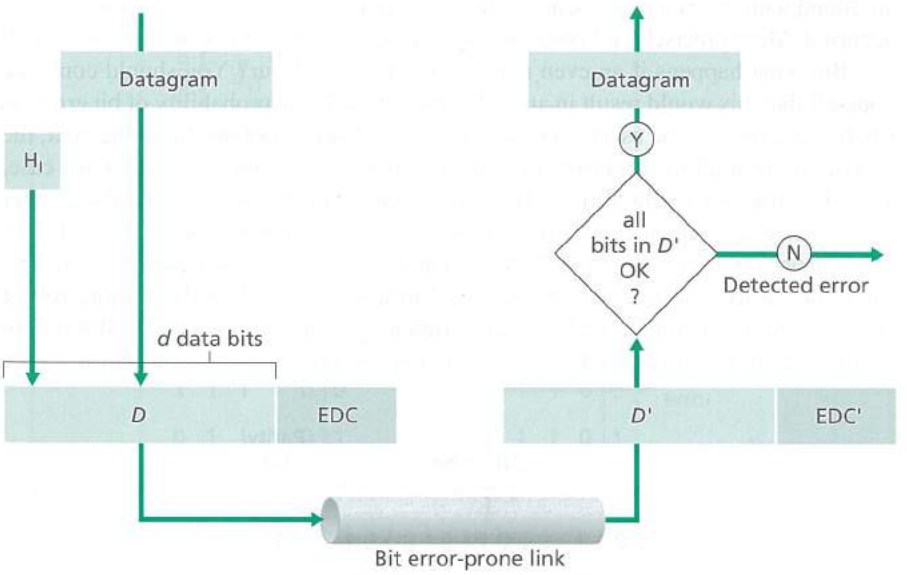


Figure 6.3 ♦ Error-detection and -correction scenario

consequence, the receiver might deliver a corrupted datagram to the network layer, or be unaware that the contents of a field in the frame's header has been corrupted. We thus want to choose an error-detection scheme that keeps the probability of such occurrences small. Generally, more sophisticated error-detection and-correction techniques (that is, those that have a smaller probability of allowing undetected bit errors) incur a larger overhead—more computation is needed to compute and transmit a larger number of error-detection and -correction bits.

Let's now examine three techniques for detecting errors in the transmitted data—parity checks (to illustrate the basic ideas behind error detection and correction), checksumming methods (which are more typically used in the transport layer), and cyclic redundancy checks (which are more typically used in the link layer in an adapter).

6.2.1 Parity Checks

Perhaps the simplest form of error detection is the use of a single **parity bit**. Suppose that the information to be sent, D in Figure 6.4, has d bits. In an even parity scheme, the sender simply includes one additional bit and chooses its value such that the total number of 1s in the $d + 1$ bits (the original information plus a parity bit) is even. For odd parity schemes, the parity bit value is chosen such that there is an odd number of 1s. Figure 6.4 illustrates an even parity scheme, with the single parity bit being stored in a separate field.

Receiver operation is also simple with a single parity bit. The receiver need only count the number of 1s in the received $d + 1$ bits. If an odd number of 1-valued bits are found with an even parity scheme, the receiver knows that at least one bit error has occurred. More precisely, it knows that some *odd* number of bit errors have occurred.

But what happens if an even number of bit errors occur? You should convince yourself that this would result in an undetected error. If the probability of bit errors is small and errors can be assumed to occur independently from one bit to the next, the probability of multiple bit errors in a packet would be extremely small. In this case, a single parity bit might suffice. However, measurements have shown that, rather than occurring independently, errors are often clustered together in "bursts." Under burst error conditions, the probability of undetected errors in a frame protected by single-bit parity can approach 50 percent [Spragins 1991]. Clearly, a more robust error-detection scheme is needed (and, fortunately, is used in practice!). But before examining error-detection schemes that are used in practice, let's consider a simple

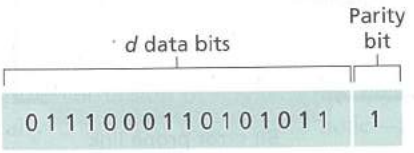


Figure 6.4 ♦ One-bit even parity

generalization of one-bit parity that will provide us with insight into error-correction techniques.

Figure 6.5 shows a two-dimensional generalization of the single-bit parity scheme. Here, the d bits in D are divided into i rows and j columns. A parity value is computed for each row and for each column. The resulting $i + j + 1$ parity bits comprise the link-layer frame's error-detection bits.

Suppose now that a single bit error occurs in the original d bits of information. With this **two-dimensional parity** scheme, the parity of both the column and the row containing the flipped bit will be in error. The receiver can thus not only *detect* the fact that a single bit error has occurred, but can use the column and row indices of the column and row with parity errors to actually identify the bit that was corrupted and *correct* that error! Figure 6.5 shows an example in which the 1-valued bit in position (2,2) is corrupted and switched to a 0—an error that is both detectable and correctable at the receiver. Although our discussion has focused on the original d bits of information, a single error in the parity bits themselves is also detectable and correctable. Two-dimensional parity can also detect (but not correct!) any combination of two errors in a packet. Other properties of the two-dimensional parity scheme are explored in the problems at the end of the chapter.

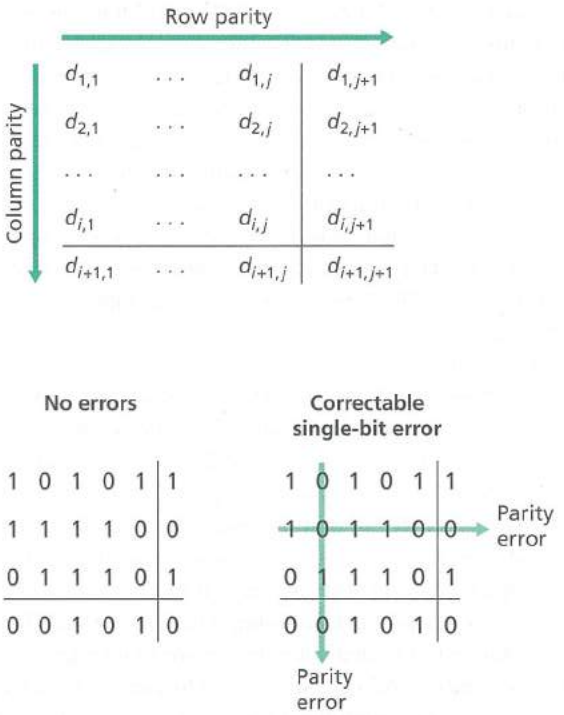


Figure 6.5 ♦ Two-dimensional even parity

The ability of the receiver to both detect and correct errors is known as **forward error correction (FEC)**. These techniques are commonly used in audio storage and playback devices such as audio CDs. In a network setting, FEC techniques can be used by themselves, or in conjunction with link-layer ARQ techniques similar to those we examined in Chapter 3. FEC techniques are valuable because they can decrease the number of sender retransmissions required. Perhaps more important, they allow for immediate correction of errors at the receiver. This avoids having to wait for the round-trip propagation delay needed for the sender to receive a NAK packet and for the retransmitted packet to propagate back to the receiver—a potentially important advantage for real-time network applications [Rubenstein 1998] or links (such as deep-space links) with long propagation delays. Research examining the use of FEC in error-control protocols includes [Biersack 1992; Nonnenmacher 1998; Byers 1998; Shacham 1990].

6.2.2 Checksumming Methods

In checksumming techniques, the d bits of data in Figure 6.4 are treated as a sequence of k-bit integers. One simple checksumming method is to simply sum these k-bit integers and use the resulting sum as the error-detection bits. The **Internet checksum** is based on this approach—bytes of data are treated as 16-bit integers and summed. The 1s complement of this sum then forms the Internet checksum that is carried in the segment header. As discussed in Section 3.3, the receiver checks the checksum by taking the 1s complement of the sum of the received data (including the checksum) and checking whether the result is all 1 bits. If any of the bits are 0, an error is indicated. RFC 1071 discusses the Internet checksum algorithm and its implementation in detail. In the TCP and UDP protocols, the Internet checksum is computed over all fields (header and data fields included). In IP the checksum is computed over the IP header (since the UDP or TCP segment has its own checksum). In other protocols, for example, XTP [Strayer 1992], one checksum is computed over the header and another checksum is computed over the entire packet.

Checksumming methods require relatively little packet overhead. For example, the checksums in TCP and UDP use only 16 bits. However, they provide relatively weak protection against errors as compared with cyclic redundancy check, which is discussed below and which is often used in the link layer. A natural question at this point is, Why is checksumming used at the transport layer and cyclic redundancy check used at the link layer? Recall that the transport layer is typically implemented in software in a host as part of the host's operating system. Because transport-layer error detection is implemented in software, it is important to have a simple and fast error-detection scheme such as checksumming. On the other hand, error detection at the link layer is implemented in dedicated hardware in adapters, which can rapidly perform the more complex CRC operations. Feldmeier [Feldmeier 1995] presents fast software implementation techniques for not only weighted checksum codes, but CRC (see below) and other codes as well.

6.2.3 Cyclic Redundancy Check (CRC)

An error-detection technique used widely in today's computer networks is based on **cyclic redundancy check (CRC) codes**. CRC codes are also known as **polynomial codes**, since it is possible to view the bit string to be sent as a polynomial whose coefficients are the 0 and 1 values in the bit string, with operations on the bit string interpreted as polynomial arithmetic.

CRC codes operate as follows. Consider the d-bit piece of data, D, that the sending node wants to send to the receiving node. The sender and receiver must first agree on an $r + 1$ bit pattern, known as a **generator**, which we will denote as G. We will require that the most significant (leftmost) bit of G be a 1. The key idea behind CRC codes is shown in Figure 6.6. For a given piece of data, D, the sender will choose r additional bits, R, and append them to D such that the resulting $d + r$ bit pattern (interpreted as a binary number) is exactly divisible by G (i.e., has no remainder) using modulo-2 arithmetic. The process of error checking with CRCs is thus simple: The receiver divides the $d + r$ received bits by G. If the remainder is nonzero, the receiver knows that an error has occurred; otherwise the data is accepted as being correct.

All CRC calculations are done in modulo-2 arithmetic without carries in addition or borrows in subtraction. This means that addition and subtraction are identical, and both are equivalent to the bitwise exclusive-or (XOR) of the operands. Thus, for example,

```
1011 XOR 0101 = 1110
1001 XOR 1101 = 0100
```

Also, we similarly have

```
1011 - 0101 = 1110
1001 - 1101 = 0100
```

Multiplication and division are the same as in base-2 arithmetic, except that any required addition or subtraction is done without carries or borrows. As in regular

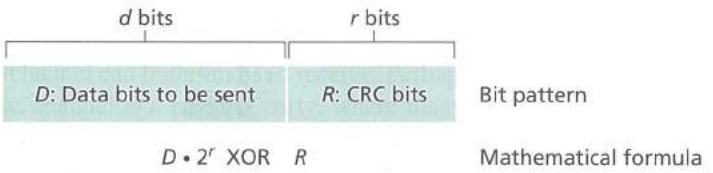


Figure 6.6 ♦ CRC

binary arithmetic, multiplication by 2^k left shifts a bit pattern by k places. Thus, given D and R, the quantity $D \cdot 2^r$ XOR R yields the $d + r$ bit pattern shown in Figure 6.6. We'll use this algebraic characterization of the $d + r$ bit pattern from Figure 6.6 in our discussion below.

Let us now turn to the crucial question of how the sender computes R. Recall that we want to find R such that there is an n such that

$$D \cdot 2^r \text{ XOR } R = nG$$

That is, we want to choose R such that G divides into $D \cdot 2^r$ XOR R without remainder. If we XOR (that is, add modulo-2, without carry) R to both sides of the above equation, we get

$$D \cdot 2^r = nG \text{ XOR } R$$

This equation tells us that if we divide $D \cdot 2^r$ by G, the value of the remainder is precisely R. In other words, we can calculate R as

$$R = \text{remainder} \frac{D \cdot 2^r}{G}$$

Figure 6.7 illustrates this calculation for the case of $D = 101110$, $d = 6$, $G = 1001$, and $r = 3$. The 9 bits transmitted in this case are 101 110 011. You should check these calculations for yourself and also check that indeed $D \cdot 2^r = 101011 \cdot G$ XOR R.

```
G                 1 0 1 0 1 1
1 0 0 1 | 1 0 1 1 1 0 0 0 0
          1 0 0 1
            1 0 1          D
            0 0 0
            1 0 1 0
            1 0 0 1
              1 1 0
              0 0 0
              1 1 0 0
              1 0 0 1
                1 0 1 0
                1 0 0 1
                  0 1 1
                    R
```

Figure 6.7 ♦ A sample CRC calculation

International standards have been defined for 8-, 12-, 16-, and 32-bit generators, G. The CRC-32 32-bit standard, which has been adopted in a number of link-level IEEE protocols, uses a generator of

$$G_{\text{CRC-32}} = 100000100110000010001110110110111$$

Each of the CRC standards can detect burst errors of fewer than $r + 1$ bits. (This means that all consecutive bit errors of r bits or fewer will be detected.) Furthermore, under appropriate assumptions, a burst of length greater than $r + 1$ bits is detected with probability $1 - 0.5^r$. Also, each of the CRC standards can detect any odd number of bit errors. See [Williams 1993] for a discussion of implementing CRC checks. The theory behind CRC codes and even more powerful codes is beyond the scope of this text. The text [Schwartz 1980] provides an excellent introduction to this topic.

6.3 Multiple Access Links and Protocols

In the introduction to this chapter, we noted that there are two types of network links: point-to-point links and broadcast links. A **point-to-point link** consists of a single sender at one end of the link and a single receiver at the other end of the link. Many link-layer protocols have been designed for point-to-point links; the point-to-point protocol (PPP) and high-level data link control (HDLC) are two such protocols. The second type of link, a **broadcast link**, can have multiple sending and receiving nodes all connected to the same, single, shared broadcast channel. The term *broadcast* is used here because when any one node transmits a frame, the channel broadcasts the frame and each of the other nodes receives a copy. Ethernet and wireless LANs are examples of broadcast link-layer technologies. In this section we'll take a step back from specific link-layer protocols and first examine a problem of central importance to the link layer: how to coordinate the access of multiple sending and receiving nodes to a shared broadcast channel—the **multiple access problem**. Broadcast channels are often used in LANs, networks that are geographically concentrated in a single building (or on a corporate or university campus). Thus, we'll look at how multiple access channels are used in LANs at the end of this section.

We are all familiar with the notion of broadcasting—television has been using it since its invention. But traditional television is a one-way broadcast (that is, one fixed node transmitting to many receiving nodes), while nodes on a computer network broadcast channel can both send and receive. Perhaps a more apt human analogy for a broadcast channel is a cocktail party, where many people gather in a large room (the air providing the broadcast medium) to talk and listen. A second good analogy is something many readers will be familiar with—a classroom—where teacher(s) and student(s) similarly share the same, single, broadcast medium. A central problem in

both scenarios is that of determining who gets to talk (that is, transmit into the channel) and when. As humans, we've evolved an elaborate set of protocols for sharing the broadcast channel:

"Give everyone a chance to speak."
"Don't speak until you are spoken to."
"Don't monopolize the conversation."
"Raise your hand if you have a question."
"Don't interrupt when someone is speaking."
"Don't fall asleep when someone is talking."

Computer networks similarly have protocols—so-called **multiple access protocols**—by which nodes regulate their transmission into the shared broadcast channel. As shown in Figure 6.8, multiple access protocols are needed in a wide variety of network settings, including both wired and wireless access networks, and satellite networks. Although technically each node accesses the broadcast channel through its adapter, in this section we will refer to the *node* as the sending and

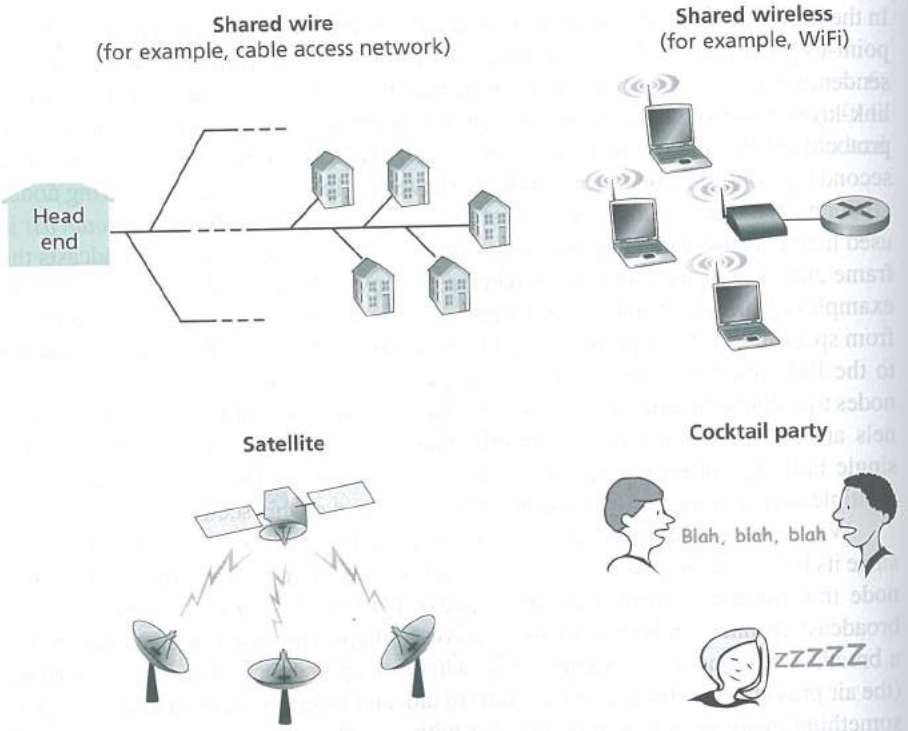


Figure 6.8 ♦ Various multiple access channels

receiving device. In practice, hundreds or even thousands of nodes can directly communicate over a broadcast channel.

Because all nodes are capable of transmitting frames, more than two nodes can transmit frames at the same time. When this happens, all of the nodes receive multiple frames at the same time; that is, the transmitted frames **collide** at all of the receivers. Typically, when there is a collision, none of the receiving nodes can make any sense of any of the frames that were transmitted; in a sense, the signals of the colliding frames become inextricably tangled together. Thus, all the frames involved in the collision are lost, and the broadcast channel is wasted during the collision interval. Clearly, if many nodes want to transmit frames frequently, many transmissions will result in collisions, and much of the bandwidth of the broadcast channel will be wasted.

In order to ensure that the broadcast channel performs useful work when multiple nodes are active, it is necessary to somehow coordinate the transmissions of the active nodes. This coordination job is the responsibility of the multiple access protocol. Over the past 40 years, thousands of papers and hundreds of PhD dissertations have been written on multiple access protocols; a comprehensive survey of the first 20 years of this body of work is [Rom 1990]. Furthermore, active research in multiple access protocols continues due to the continued emergence of new types of links, particularly new wireless links.

Over the years, dozens of multiple access protocols have been implemented in a variety of link-layer technologies. Nevertheless, we can classify just about any multiple access protocol as belonging to one of three categories: **channel partitioning protocols**, **random access protocols**, and **taking-turns protocols**. We'll cover these categories of multiple access protocols in the following three subsections.

Let's conclude this overview by noting that, ideally, a multiple access protocol for a broadcast channel of rate R bits per second should have the following desirable characteristics:

1. When only one node has data to send, that node has a throughput of R bps.
2. When M nodes have data to send, each of these nodes has a throughput of R/M bps. This need not necessarily imply that each of the M nodes always has an instantaneous rate of R/M, but rather that each node should have an average transmission rate of R/M over some suitably defined interval of time.
3. The protocol is decentralized; that is, there is no master node that represents a single point of failure for the network.
4. The protocol is simple, so that it is inexpensive to implement.

6.3.1 Channel Partitioning Protocols

Recall from our early discussion back in Section 1.3 that time-division multiplexing (TDM) and frequency-division multiplexing (FDM) are two techniques that can

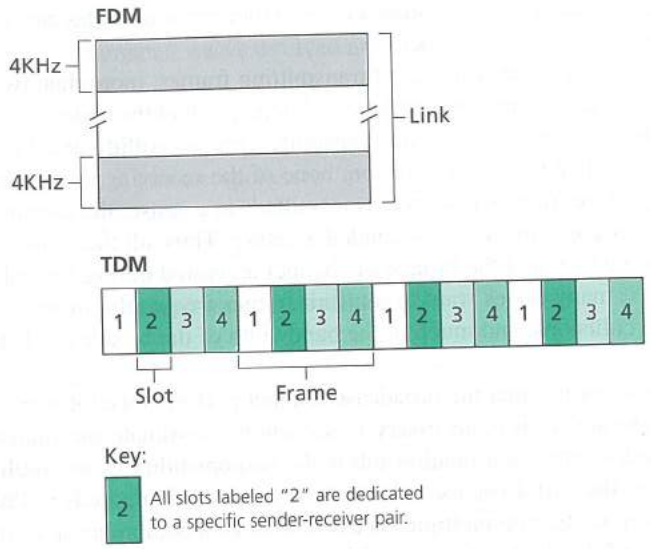


Figure 6.9 ♦ A four-node TDM and FDM example

be used to partition a broadcast channel's bandwidth among all nodes sharing that channel. As an example, suppose the channel supports N nodes and that the transmission rate of the channel is R bps. TDM divides time into **time frames** and further divides each time frame into N **time slots**. (The TDM time frame should not be confused with the link-layer unit of data exchanged between sending and receiving adapters, which is also called a frame. In order to reduce confusion, in this subsection we'll refer to the link-layer unit of data exchanged as a packet.) Each time slot is then assigned to one of the N nodes. Whenever a node has a packet to send, it transmits the packet's bits during its assigned time slot in the revolving TDM frame. Typically, slot sizes are chosen so that a single packet can be transmitted during a slot time. Figure 6.9 shows a simple four-node TDM example. Returning to our cocktail party analogy, a TDM-regulated cocktail party would allow one partygoer to speak for a fixed period of time, then allow another partygoer to speak for the same amount of time, and so on. Once everyone had had a chance to talk, the pattern would repeat.

TDM is appealing because it eliminates collisions and is perfectly fair: Each node gets a dedicated transmission rate of R/N bps during each frame time. However, it has two major drawbacks. First, a node is limited to an average rate of R/N bps even when it is the only node with packets to send. A second drawback is that a node must always wait for its turn in the transmission sequence—again, even when it is the only node with a frame to send. Imagine the partygoer who is the only one with anything to say (and imagine that this is the even rarer circumstance where everyone

wants to hear what that one person has to say). Clearly, TDM would be a poor choice for a multiple access protocol for this particular party.

While TDM shares the broadcast channel in time, FDM divides the R bps channel into different frequencies (each with a bandwidth of R/N) and assigns each frequency to one of the N nodes. FDM thus creates N smaller channels of R/N bps out of the single, larger R bps channel. FDM shares both the advantages and drawbacks of TDM. It avoids collisions and divides the bandwidth fairly among the N nodes. However, FDM also shares a principal disadvantage with TDM—a node is limited to a bandwidth of R/N, even when it is the only node with packets to send.

A third channel partitioning protocol is **code division multiple access (CDMA)**. While TDM and FDM assign time slots and frequencies, respectively, to the nodes, CDMA assigns a different *code* to each node. Each node then uses its unique code to encode the data bits it sends. If the codes are chosen carefully, CDMA networks have the wonderful property that different nodes can transmit *simultaneously* and yet have their respective receivers correctly receive a sender's encoded data bits (assuming the receiver knows the sender's code) in spite of interfering transmissions by other nodes. CDMA has been used in military systems for some time (due to its anti-jamming properties) and now has widespread civilian use, particularly in cellular telephony. Because CDMA's use is so tightly tied to wireless channels, we'll save our discussion of the technical details of CDMA until Chapter 7. For now, it will suffice to know that CDMA codes, like time slots in TDM and frequencies in FDM, can be allocated to the multiple access channel users.

6.3.2 Random Access Protocols

The second broad class of multiple access protocols are random access protocols. In a random access protocol, a transmitting node always transmits at the full rate of the channel, namely, R bps. When there is a collision, each node involved in the collision repeatedly retransmits its frame (that is, packet) until its frame gets through without a collision. But when a node experiences a collision, it doesn't necessarily retransmit the frame right away. *Instead it waits a random delay before retransmitting the frame*. Each node involved in a collision chooses independent random delays. Because the random delays are independently chosen, it is possible that one of the nodes will pick a delay that is sufficiently less than the delays of the other colliding nodes and will therefore be able to sneak its frame into the channel without a collision.

There are dozens if not hundreds of random access protocols described in the literature [Rom 1990; Bertsekas 1991]. In this section we'll describe a few of the most commonly used random access protocols—the ALOHA protocols [Abramson 1970; Abramson 1985; Abramson 2009] and the carrier sense multiple access (CSMA) protocols [Kleinrock 1975b]. Ethernet [Metcalfe 1976] is a popular and widely deployed CSMA protocol.

Slotted ALOHA

Let's begin our study of random access protocols with one of the simplest random access protocols, the slotted ALOHA protocol. In our description of slotted ALOHA, we assume the following:

- All frames consist of exactly L bits.
- Time is divided into slots of size L/R seconds (that is, a slot equals the time to transmit one frame).
- Nodes start to transmit frames only at the beginnings of slots.
- The nodes are synchronized so that each node knows when the slots begin.
- If two or more frames collide in a slot, then all the nodes detect the collision event before the slot ends.

Let p be a probability, that is, a number between 0 and 1. The operation of slotted ALOHA in each node is simple:

- When the node has a fresh frame to send, it waits until the beginning of the next slot and transmits the entire frame in the slot.
- If there isn't a collision, the node has successfully transmitted its frame and thus need not consider retransmitting the frame. (The node can prepare a new frame for transmission, if it has one.)
- If there is a collision, the node detects the collision before the end of the slot. The node retransmits its frame in each subsequent slot with probability p until the frame is transmitted without a collision.

By retransmitting with probability p, we mean that the node effectively tosses a biased coin; the event heads corresponds to "retransmit," which occurs with probability p. The event tails corresponds to "skip the slot and toss the coin again in the next slot"; this occurs with probability $(1 - p)$. All nodes involved in the collision toss their coins independently.

Slotted ALOHA would appear to have many advantages. Unlike channel partitioning, slotted ALOHA allows a node to transmit continuously at the full rate, R, when that node is the only active node. (A node is said to be active if it has frames to send.) Slotted ALOHA is also highly decentralized, because each node detects collisions and independently decides when to retransmit. (Slotted ALOHA does, however, require the slots to be synchronized in the nodes; shortly we'll discuss an unslotted version of the ALOHA protocol, as well as CSMA protocols, none of which require such synchronization.) Slotted ALOHA is also an extremely simple protocol.

Slotted ALOHA works well when there is only one active node, but how efficient is it when there are multiple active nodes? There are two possible efficiency

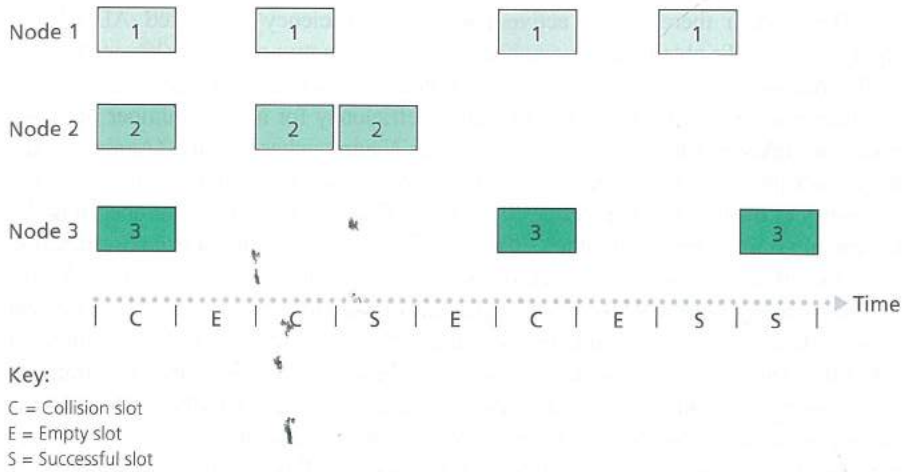


Figure 6.10 ♦ Nodes 1, 2, and 3 collide in the first slot. Node 2 finally succeeds in the fourth slot, node 1 in the eighth slot, and node 3 in the ninth slot

concerns here. First, as shown in Figure 6.10, when there are multiple active nodes, a certain fraction of the slots will have collisions and will therefore be "wasted." The second concern is that another fraction of the slots will be *empty* because all active nodes refrain from transmitting as a result of the probabilistic transmission policy. The only "unwasted" slots will be those in which exactly one node transmits. A slot in which exactly one node transmits is said to be a **successful slot**. The **efficiency** of a slotted multiple access protocol is defined to be the long-run fraction of successful slots in the case when there are a large number of active nodes, each always having a large number of frames to send. Note that if no form of access control were used, and each node were to immediately retransmit after each collision, the efficiency would be zero. Slotted ALOHA clearly increases the efficiency beyond zero, but by how much?

We now proceed to outline the derivation of the maximum efficiency of slotted ALOHA. To keep this derivation simple, let's modify the protocol a little and assume that each node attempts to transmit a frame in each slot with probability p. (That is, we assume that each node always has a frame to send and that the node transmits with probability p for a fresh frame as well as for a frame that has already suffered a collision.) Suppose there are N nodes. Then the probability that a given slot is a successful slot is the probability that one of the nodes transmits and that the remaining $N - 1$ nodes do not transmit. The probability that a given node transmits is p; the probability that the remaining nodes do not transmit is $(1 - p)^{N-1}$. Therefore the probability a given node has a success is $p(1 - p)^{N-1}$. Because there are N nodes, the probability that any one of the N nodes has a success is $Np(1 - p)^{N-1}$.

Thus, when there are N active nodes, the efficiency of slotted ALOHA is $Np(1 - p)^{N-1}$. To obtain the *maximum* efficiency for N active nodes, we have to find the p^* that maximizes this expression. (See the homework problems for a general outline of this derivation.) And to obtain the maximum efficiency for a large number of active nodes, we take the limit of $Np^*(1 - p^*)^{N-1}$ as N approaches infinity. (Again, see the homework problems.) After performing these calculations, we'll find that the maximum efficiency of the protocol is given by $1/e = 0.37$. That is, when a large number of nodes have many frames to transmit, then (at best) only 37 percent of the slots do useful work. Thus the effective transmission rate of the channel is not R bps but only 0.37 R bps! A similar analysis also shows that 37 percent of the slots go empty and 26 percent of slots have collisions. Imagine the poor network administrator who has purchased a 100-Mbps slotted ALOHA system, expecting to be able to use the network to transmit data among a large number of users at an aggregate rate of, say, 80 Mbps! Although the channel is capable of transmitting a given frame at the full channel rate of 100 Mbps, in the long run, the successful throughput of this channel will be less than 37 Mbps.

ALOHA

The slotted ALOHA protocol required that all nodes synchronize their transmissions to start at the beginning of a slot. The first ALOHA protocol [Abramson 1970] was actually an unslotted, fully decentralized protocol. In pure ALOHA, when a frame first arrives (that is, a network-layer datagram is passed down from the network layer at the sending node), the node immediately transmits the frame in its entirety into the broadcast channel. If a transmitted frame experiences a collision with one or more other transmissions, the node will then immediately (after completely transmitting its collided frame) retransmit the frame with probability p. Otherwise, the node waits for a frame transmission time. After this wait, it then transmits the frame with probability p, or waits (remaining idle) for another frame time with probability $1 - p$.

To determine the maximum efficiency of pure ALOHA, we focus on an individual node. We'll make the same assumptions as in our slotted ALOHA analysis and take the frame transmission time to be the unit of time. At any given time, the probability that a node is transmitting a frame is p. Suppose this frame begins transmission at time t_0. As shown in Figure 6.11, in order for this frame to be successfully transmitted, no other nodes can begin their transmission in the interval of time $[t_0 - 1, t_0]$. Such a transmission would overlap with the beginning of the transmission of node i's frame. The probability that all other nodes do not begin a transmission in this interval is $(1 - p)^{N-1}$. Similarly, no other node can begin a transmission while node i is transmitting, as such a transmission would overlap with the latter part of node i's transmission. The probability that all other nodes do not begin a transmission in this interval is also $(1 - p)^{N-1}$. Thus, the probability that a given node has a successful transmission is $p(1 - p)^{2(N-1)}$. By taking limits as in the slotted ALOHA case, we find that the maximum efficiency of the pure ALOHA protocol is only $1/(2e)$—exactly half that of slotted ALOHA. This then is the price to be paid for a fully decentralized ALOHA protocol.

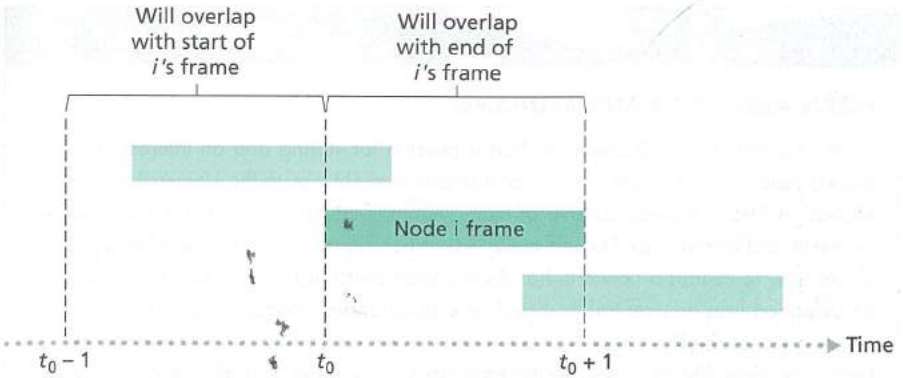


Figure 6.11 ♦ Interfering transmissions in pure ALOHA

Carrier Sense Multiple Access (CSMA)

In both slotted and pure ALOHA, a node's decision to transmit is made independently of the activity of the other nodes attached to the broadcast channel. In particular, a node neither pays attention to whether another node happens to be transmitting when it begins to transmit, nor stops transmitting if another node begins to interfere with its transmission. In our cocktail party analogy, ALOHA protocols are quite like a boorish partygoer who continues to chatter away regardless of whether other people are talking. As humans, we have human protocols that allow us not only to behave with more civility, but also to decrease the amount of time spent "colliding" with each other in conversation and, consequently, to increase the amount of data we exchange in our conversations. Specifically, there are two important rules for polite human conversation:

- *Listen before speaking.* If someone else is speaking, wait until they are finished. In the networking world, this is called **carrier sensing**—a node listens to the channel before transmitting. If a frame from another node is currently being transmitted into the channel, a node then waits until it detects no transmissions for a short amount of time and then begins transmission.
- *If someone else begins talking at the same time, stop talking.* In the networking world, this is called **collision detection**—a transmitting node listens to the channel while it is transmitting. If it detects that another node is transmitting an interfering frame, it stops transmitting and waits a random amount of time before repeating the sense-and-transmit-when-idle cycle.

These two rules are embodied in the family of **carrier sense multiple access (CSMA)** and **CSMA with collision detection (CSMA/CD)** protocols [Kleinrock 1975b; Metcalfe 1976; Lam 1980; Rom 1990]. Many variations on CSMA and

CASE HISTORY

NORM ABRAMSON AND ALOHANET

Norm Abramson, a PhD engineer, had a passion for surfing and an interest in packet switching. This combination of interests brought him to the University of Hawaii in 1969. Hawaii consists of many mountainous islands, making it difficult to install and operate land-based networks. When not surfing, Abramson thought about how to design a network that does packet switching over radio. The network he designed had one central host and several secondary nodes scattered over the Hawaiian Islands. The network had two channels, each using a different frequency band. The downlink channel broadcasted packets from the central host to the secondary hosts; and the upstream channel sent packets from the secondary hosts to the central host. In addition to sending informational packets, the central host also sent on the downstream channel an acknowledgment for each packet successfully received from the secondary hosts.

Because the secondary hosts transmitted packets in a decentralized fashion, collisions on the upstream channel inevitably occurred. This observation led Abramson to devise the pure ALOHA protocol, as described in this chapter. In 1970, with continued funding from ARPA, Abramson connected his ALOHAnet to the ARPAnet. Abramson's work is important not only because it was the first example of a radio packet network, but also because it inspired Bob Metcalfe. A few years later, Metcalfe modified the ALOHA protocol to create the CSMA/CD protocol and the Ethernet LAN.

CSMA/CD have been proposed. Here, we'll consider a few of the most important, and fundamental, characteristics of CSMA and CSMA/CD.

The first question that you might ask about CSMA is why, if all nodes perform carrier sensing, do collisions occur in the first place? After all, a node will refrain from transmitting whenever it senses that another node is transmitting. The answer to the question can best be illustrated using space-time diagrams [Molle 1987]. Figure 6.12 shows a space-time diagram of four nodes (A, B, C, D) attached to a linear broadcast bus. The horizontal axis shows the position of each node in space; the vertical axis represents time.

At time t_0, node B senses the channel is idle, as no other nodes are currently transmitting. Node B thus begins transmitting, with its bits propagating in both directions along the broadcast medium. The downward propagation of B's bits in Figure 6.12 with increasing time indicates that a nonzero amount of time is needed for B's bits actually to propagate (albeit at near the speed of light) along the broadcast medium. At time t_1 ($t_1 > t_0$), node D has a frame to send. Although node B is currently transmitting at time t_1, the bits being transmitted by B have yet to reach D, and thus D senses

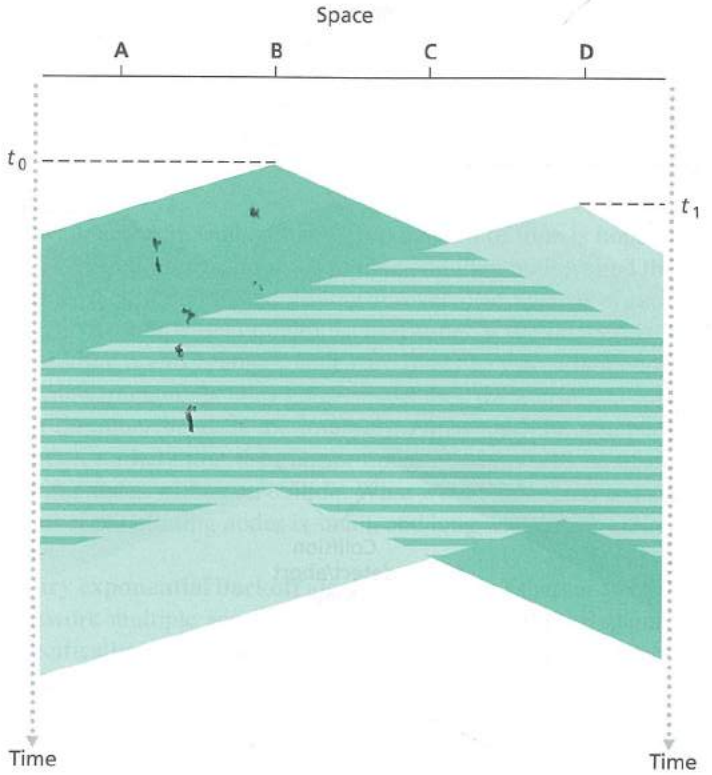


Figure 6.12 ♦ Space-time diagram of two CSMA nodes with colliding transmissions

the channel idle at t_1. In accordance with the CSMA protocol, D thus begins transmitting its frame. A short time later, B's transmission begins to interfere with D's transmission at D. From Figure 6.12, it is evident that the end-to-end **channel propagation delay** of a broadcast channel—the time it takes for a signal to propagate from one of the nodes to another—will play a crucial role in determining its performance. The longer this propagation delay, the larger the chance that a carrier-sensing node is not yet able to sense a transmission that has already begun at another node in the network.

Carrier Sense Multiple Access with Collision Dection (CSMA/CD)

In Figure 6.12, nodes do not perform collision detection; both B and D continue to transmit their frames in their entirety even though a collision has occurred. When a node performs collision detection, it ceases transmission as soon as it detects a collision. Figure 6.13 shows the same scenario as in Figure 6.12, except that the two

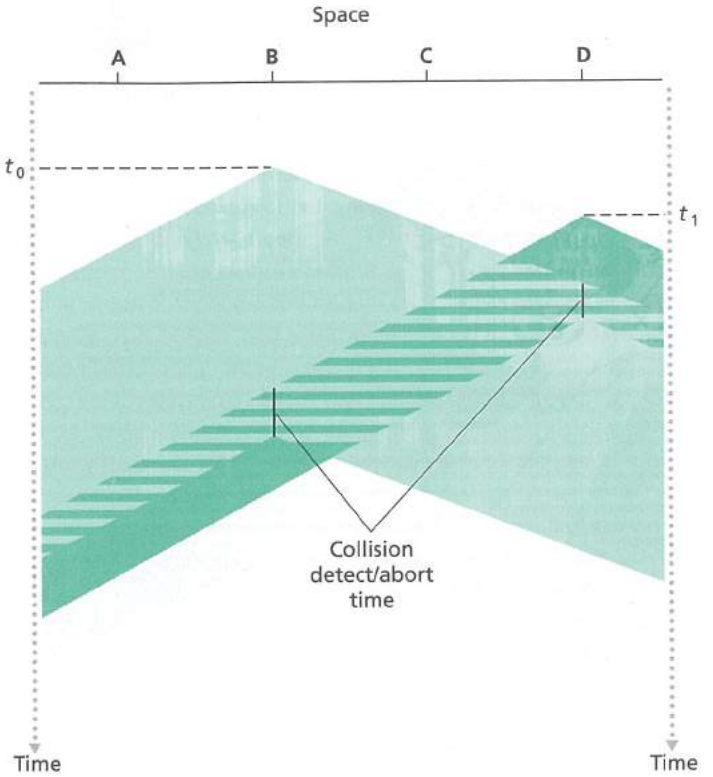


Figure 6.13 ♦ CSMA with collision detection

nodes each abort their transmission a short time after detecting a collision. Clearly, adding collision detection to a multiple access protocol will help protocol performance by not transmitting a useless, damaged (by interference with a frame from another node) frame in its entirety.

Before analyzing the CSMA/CD protocol, let us now summarize its operation from the perspective of an adapter (in a node) attached to a broadcast channel:

1. The adapter obtains a datagram from the network layer, prepares a link-layer frame, and puts the frame adapter buffer.
2. If the adapter senses that the channel is idle (that is, there is no signal energy entering the adapter from the channel), it starts to transmit the frame. If, on the other hand, the adapter senses that the channel is busy, it waits until it senses no signal energy and then starts to transmit the frame.
3. While transmitting, the adapter monitors for the presence of signal energy coming from other adapters using the broadcast channel.

4. If the adapter transmits the entire frame without detecting signal energy from other adapters, the adapter is finished with the frame. If, on the other hand, the adapter detects signal energy from other adapters while transmitting, it aborts the transmission (that is, it stops transmitting its frame).
5. After aborting, the adapter waits a random amount of time and then returns to step 2.

The need to wait a random (rather than fixed) amount of time is hopefully clear—if two nodes transmitted frames at the same time and then both waited the same fixed amount of time, they'd continue colliding forever. But what is a good interval of time from which to choose the random backoff time? If the interval is large and the number of colliding nodes is small, nodes are likely to wait a large amount of time (with the channel remaining idle) before repeating the sense-and-transmit-when-idle step. On the other hand, if the interval is small and the number of colliding nodes is large, it's likely that the chosen random values will be nearly the same, and transmitting nodes will again collide. What we'd like is an interval that is short when the number of colliding nodes is small, and long when the number of colliding nodes is large.

The **binary exponential backoff** algorithm, used in Ethernet as well as in DOCSIS cable network multiple access protocols [DOCSIS 2011], elegantly solves this problem. Specifically, when transmitting a frame that has already experienced n collisions, a node chooses the value of K at random from $\{0,1,2, \ldots . 2^n-1\}$. Thus, the more collisions experienced by a frame, the larger the interval from which K is chosen. For Ethernet, the actual amount of time a node waits is $K \cdot 512$ bit times (i.e., K times the amount of time needed to send 512 bits into the Ethernet) and the maximum value that n can take is capped at 10.

Let's look at an example. Suppose that a node attempts to transmit a frame for the first time and while transmitting it detects a collision. The node then chooses $K = 0$ with probability 0.5 or chooses $K = 1$ with probability 0.5. If the node chooses $K = 0$, then it immediately begins sensing the channel. If the node chooses $K = 1$, it waits 512 bit times (e.g., 5.12 microseconds for a 100 Mbps Ethernet) before beginning the sense-and-transmit-when-idle cycle. After a second collision, K is chosen with equal probability from $\{0,1,2,3\}$. After three collisions, K is chosen with equal probability from $\{0,1,2,3,4,5,6,7\}$. After 10 or more collisions, K is chosen with equal probability from $\{0,1,2,\ldots, 1023\}$. Thus, the size of the sets from which K is chosen grows exponentially with the number of collisions; for this reason this algorithm is referred to as binary exponential backoff.

We also note here that each time a node prepares a new frame for transmission, it runs the CSMA/CD algorithm, not taking into account any collisions that may have occurred in the recent past. So it is possible that a node with a new frame will immediately be able to sneak in a successful transmission while several other nodes are in the exponential backoff state.

CSMA/CD Efficiency

When only one node has a frame to send, the node can transmit at the full channel rate (e.g., for Ethernet typical rates are 10 Mbps, 100 Mbps, or 1 Gbps). However, if many nodes have frames to transmit, the effective transmission rate of the channel can be much less. We define the **efficiency of CSMA/CD** to be the long-run fraction of time during which frames are being transmitted on the channel without collisions when there is a large number of active nodes, with each node having a large number of frames to send. In order to present a closed-form approximation of the efficiency of Ethernet, let d_{prop} denote the maximum time it takes signal energy to propagate between any two adapters. Let d_{trans} be the time to transmit a maximum-size frame (approximately 1.2 msecs for a 10 Mbps Ethernet). A derivation of the efficiency of CSMA/CD is beyond the scope of this book (see [Lam 1980] and [Bertsekas 1991]). Here we simply state the following approximation:

$$\text{Efficiency} = \frac{1}{1 + 5d_{prop}/d_{trans}}$$

We see from this formula that as d_{prop} approaches 0, the efficiency approaches 1. This matches our intuition that if the propagation delay is zero, colliding nodes will abort immediately without wasting the channel. Also, as d_{trans} becomes very large, efficiency approaches 1. This is also intuitive because when a frame grabs the channel, it will hold on to the channel for a very long time; thus, the channel will be doing productive work most of the time.

6.3.3 Taking-Turns Protocols

Recall that two desirable properties of a multiple access protocol are (1) when only one node is active, the active node has a throughput of R bps, and (2) when M nodes are active, then each active node has a throughput of nearly R/M bps. The ALOHA and CSMA protocols have this first property but not the second. This has motivated researchers to create another class of protocols—the **taking-turns protocols**. As with random access protocols, there are dozens of taking-turns protocols, and each one of these protocols has many variations. We'll discuss two of the more important protocols here. The first one is the **polling protocol**. The polling protocol requires one of the nodes to be designated as a master node. The master node **polls** each of the nodes in a round-robin fashion. In particular, the master node first sends a message to node 1, saying that it (node 1) can transmit up to some maximum number of frames. After node 1 transmits some frames, the master node tells node 2 it (node 2) can transmit up to the maximum number of frames. (The master node can determine when a node has finished sending its frames by observing the lack of a signal on the channel.) The procedure continues in this manner, with the master node polling each of the nodes in a cyclic manner.

The polling protocol eliminates the collisions and empty slots that plague random access protocols. This allows polling to achieve a much higher efficiency. But

it also has a few drawbacks. The first drawback is that the protocol introduces a polling delay—the amount of time required to notify a node that it can transmit. If, for example, only one node is active, then the node will transmit at a rate less than R bps, as the master node must poll each of the inactive nodes in turn each time the active node has sent its maximum number of frames. The second drawback, which is potentially more serious, is that if the master node fails, the entire channel becomes inoperative. The 802.15 protocol and the Bluetooth protocol we will study in Section 6.3 are examples of polling protocols.

The second taking-turns protocol is the **token-passing protocol**. In this protocol there is no master node. A small, special-purpose frame known as a **token** is exchanged among the nodes in some fixed order. For example, node 1 might always send the token to node 2, node 2 might always send the token to node 3, and node N might always send the token to node 1. When a node receives a token, it holds onto the token only if it has some frames to transmit; otherwise, it immediately forwards the token to the next node. If a node does have frames to transmit when it receives the token, it sends up to a maximum number of frames and then forwards the token to the next node. Token passing is decentralized and highly efficient. But it has its problems as well. For example, the failure of one node can crash the entire channel. Or if a node accidentally neglects to release the token, then some recovery procedure must be invoked to get the token back in circulation. Over the years many token-passing protocols have been developed, including the fiber distributed data interface (FDDI) protocol [Jain 1994] and the IEEE 802.5 token ring protocol [IEEE 802.5 2012], and each one had to address these as well as other sticky issues.

6.3.4 DOCSIS: The Link-Layer Protocol for Cable Internet Access

In the previous three subsections, we've learned about three broad classes of multiple access protocols: channel partitioning protocols, random access protocols, and taking turns protocols. A cable access network will make for an excellent case study here, as we'll find aspects of *each* of these three classes of multiple access protocols with the cable access network!

Recall from Section 1.2.1 that a cable access network typically connects several thousand residential cable modems to a cable modem termination system (CMTS) at the cable network headend. The Data-Over-Cable Service Interface Specifications (DOCSIS) [DOCSIS 2011] specifies the cable data network architecture and its protocols. DOCSIS uses FDM to divide the downstream (CMTS to modem) and upstream (modem to CMTS) network segments into multiple frequency channels. Each downstream channel is 6 MHz wide, with a maximum throughput of approximately 40 Mbps per channel (although this data rate is seldom seen at a cable modem in practice); each upstream channel has a maximum channel width of 6.4 MHz, and a maximum upstream throughput of approximately 30 Mbps. Each upstream and

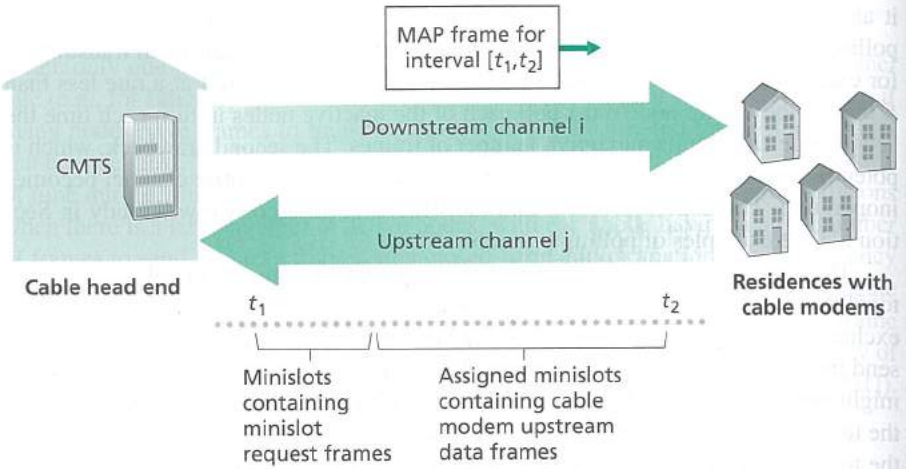


Figure 6.14 ♦ Upstream and downstream channels between CMTS and cable modems

downstream channel is a broadcast channel. Frames transmitted on the downstream channel by the CMTS are received by all cable modems receiving that channel; since there is just a single CMTS transmitting into the downstream channel, however, there is no multiple access problem. The upstream direction, however, is more interesting and technically challenging, since multiple cable modems share the same upstream channel (frequency) to the CMTS, and thus collisions can potentially occur.

As illustrated in Figure 6.14, each upstream channel is divided into intervals of time (TDM-like), each containing a sequence of mini-slots during which cable modems can transmit to the CMTS. The CMTS explicitly grants permission to individual cable modems to transmit during specific mini-slots. The CMTS accomplishes this by sending a control message known as a MAP message on a downstream channel to specify which cable modem (with data to send) can transmit during which mini-slot for the interval of time specified in the control message. Since mini-slots are explicitly allocated to cable modems, the CMTS can ensure there are no colliding transmissions during a mini-slot.

But how does the CMTS know which cable modems have data to send in the first place? This is accomplished by having cable modems send mini-slot-request frames to the CMTS during a special set of interval mini-slots that are dedicated for this purpose, as shown in Figure 6.14. These mini-slot-request frames are transmitted in a random access manner and so may collide with each other. A cable modem can neither sense whether the upstream channel is busy nor detect collisions. Instead, the cable modem infers that its mini-slot-request frame experienced a collision if it does not receive a response to the requested allocation in the next downstream control message. When a collision is inferred, a cable modem uses binary exponential

backoff to defer the retransmission of its mini-slot-request frame to a future time slot. When there is little traffic on the upstream channel, a cable modem may actually transmit data frames during slots nominally assigned for mini-slot-request frames (and thus avoid having to wait for a mini-slot assignment).

A cable access network thus serves as a terrific example of multiple access protocols in action—FDM, TDM, random access, and centrally allocated time slots all within one network!

6.4 Switched Local Area Networks

Having covered broadcast networks and multiple access protocols in the previous section, let's turn our attention next to switched local networks. Figure 6.15 shows a switched local network connecting three departments, two servers and a router with four switches. Because these switches operate at the link layer, they switch link-layer frames (rather than network-layer datagrams), don't recognize network-layer addresses, and don't use routing algorithms like RIP or OSPF to determine

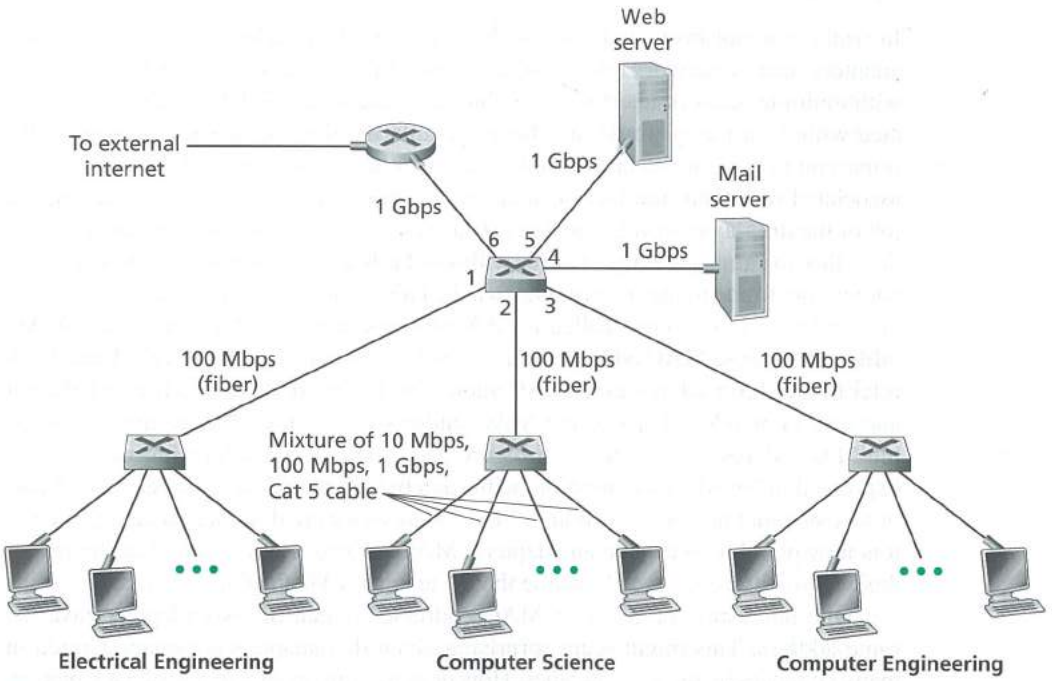


Figure 6.15 ♦ An institutional network connected together by four switches

paths through the network of layer-2 switches. Instead of using IP addresses, we will soon see that they use link-layer addresses to forward link-layer frames through the network of switches. We'll begin our study of switched LANs by first covering link-layer addressing (Section 6.4.1). We then examine the celebrated Ethernet protocol (Section 6.5.2). After examining link-layer addressing and Ethernet, we'll look at how link-layer switches operate (Section 6.4.3), and then see (Section 6.4.4) how these switches are often used to build large-scale LANs.

6.4.1 Link-Layer Addressing and ARP

Hosts and routers have link-layer addresses. Now you might find this surprising, recalling from Chapter 4 that hosts and routers have network-layer addresses as well. You might be asking, why in the world do we need to have addresses at both the network and link layers? In addition to describing the syntax and function of the link-layer addresses, in this section we hope to shed some light on why the two layers of addresses are useful and, in fact, indispensable. We'll also cover the Address Resolution Protocol (ARP), which provides a mechanism to translate IP addresses to link-layer addresses.

MAC Addresses

In truth, it is not hosts and routers that have link-layer addresses but rather their adapters (that is, network interfaces) that have link-layer addresses. A host or router with multiple network interfaces will thus have multiple link-layer addresses associated with it, just as it would also have multiple IP addresses associated with it. It's important to note, however, that link-layer switches do not have link-layer addresses associated with their interfaces that connect to hosts and routers. This is because the job of the link-layer switch is to carry datagrams between hosts and routers; a switch does this job transparently, that is, without the host or router having to explicitly address the frame to the intervening switch. This is illustrated in Figure 6.16. A link-layer address is variously called a **LAN address**, a **physical address**, or a **MAC address**. Because MAC address seems to be the most popular term, we'll henceforth refer to link-layer addresses as MAC addresses. For most LANs (including Ethernet and 802.11 wireless LANs), the MAC address is 6 bytes long, giving 2^{48} possible MAC addresses. As shown in Figure 6.16, these 6-byte addresses are typically expressed in hexadecimal notation, with each byte of the address expressed as a pair of hexadecimal numbers. Although MAC addresses were designed to be permanent, it is now possible to change an adapter's MAC address via software. For the rest of this section, however, we'll assume that an adapter's MAC address is fixed.

One interesting property of MAC addresses is that no two adapters have the same address. This might seem surprising given that adapters are manufactured in many countries by many companies. How does a company manufacturing adapters in Taiwan make sure that it is using different addresses from a company manufacturing

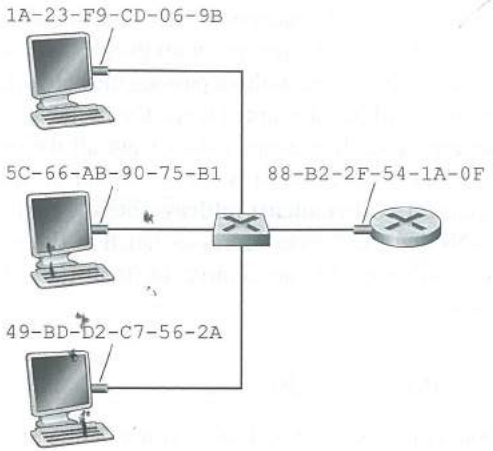


Figure 6.16 ♦ Each interface connected to a LAN has a unique MAC address

adapters in Belgium? The answer is that the IEEE manages the MAC address space. In particular, when a company wants to manufacture adapters, it purchases a chunk of the address space consisting of 2^{24} addresses for a nominal fee. IEEE allocates the chunk of 2^{24} addresses by fixing the first 24 bits of a MAC address and letting the company create unique combinations of the last 24 bits for each adapter.

An adapter's MAC address has a flat structure (as opposed to a hierarchical structure) and doesn't change no matter where the adapter goes. A laptop with an Ethernet interface always has the same MAC address, no matter where the computer goes. A smartphone with an 802.11 interface always has the same MAC address, no matter where the smartphone goes. Recall that, in contrast, IP addresses have a hierarchical structure (that is, a network part and a host part), and a host's IP addresses needs to be changed when the host moves, i.e., changes the network to which it is attached. An adapter's MAC address is analogous to a person's social security number, which also has a flat addressing structure and which doesn't change no matter where the person goes. An IP address is analogous to a person's postal address, which is hierarchical and which must be changed whenever a person moves. Just as a person may find it useful to have both a postal address and a social security number, it is useful for a host and router interfaces to have both a network-layer address and a MAC address.

When an adapter wants to send a frame to some destination adapter, the sending adapter inserts the destination adapter's MAC address into the frame and then sends the frame into the LAN. As we will soon see, a switch occasionally broadcasts an incoming frame onto all of its interfaces. We'll see in Chapter 7 that 802.11 also broadcasts frames. Thus, an adapter may receive a frame that isn't addressed to it. Thus, when an adapter receives a frame, it will check to see whether the destination MAC address

in the frame matches its own MAC address. If there is a match, the adapter extracts the enclosed datagram and passes the datagram up the protocol stack. If there isn't a match, the adapter discards the frame, without passing the network-layer datagram up. Thus, the destination only will be interrupted when the frame is received.

However, sometimes a sending adapter *does* want all the other adapters on the LAN to receive and *process* the frame it is about to send. In this case, the sending adapter inserts a special MAC **broadcast address** into the destination address field of the frame. For LANs that use 6-byte addresses (such as Ethernet and 802.11), the broadcast address is a string of 48 consecutive 1s (that is, FF-FF-FF-FF-FF-FF in hexadecimal notation).

Address Resolution Protocol (ARP)

Because there are both network-layer addresses (for example, Internet IP addresses) and link-layer addresses (that is, MAC addresses), there is a need to translate between them. For the Internet, this is the job of the **Address Resolution Protocol (ARP)** [RFC 826].

To understand the need for a protocol such as ARP, consider the network shown in Figure 6.17. In this simple example, each host and router has a single IP address and single MAC address. As usual, IP addresses are shown in dotted-decimal

PRINCIPLES IN PRACTICE

KEEPING THE LAYERS INDEPENDENT

There are several reasons why hosts and router interfaces have MAC addresses in addition to network-layer addresses. First, LANs are designed for arbitrary network-layer protocols, not just for IP and the Internet. If adapters were assigned IP addresses rather than "neutral" MAC addresses, then adapters would not easily be able to support other network-layer protocols (for example, IPX or DECnet). Second, if adapters were to use network-layer addresses instead of MAC addresses, the network-layer address would have to be stored in the adapter RAM and reconfigured every time the adapter was moved (or powered up). Another option is to not use any addresses in the adapters and have each adapter pass the data (typically, an IP datagram) of each frame it receives up the protocol stack. The network layer could then check for a matching network-layer address. One problem with this option is that the host would be interrupted by every frame sent on the LAN, including by frames that were destined for other hosts on the same broadcast LAN. In summary, in order for the layers to be largely independent building blocks in a network architecture, different layers need to have their own addressing scheme. We have now seen three types of addresses: host names for the application layer, IP addresses for the network layer, and MAC addresses for the link layer.

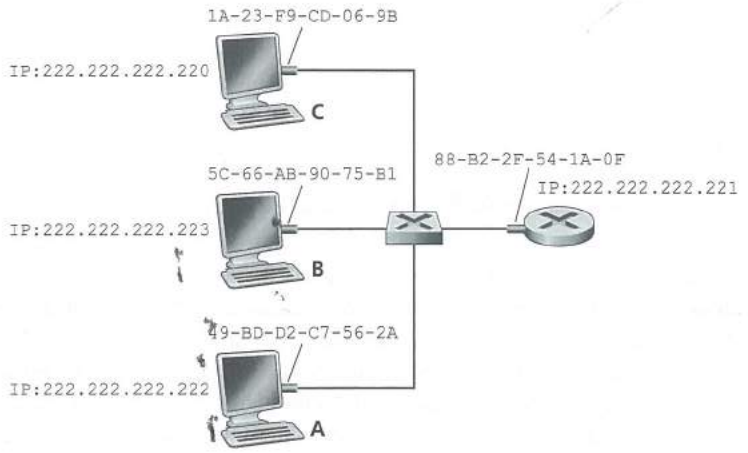


Figure 6.17 ♦ Each interface on a LAN has an IP address and a MAC address

notation and MAC addresses are shown in hexadecimal notation. For the purposes of this discussion, we will assume in this section that the switch broadcasts all frames; that is, whenever a switch receives a frame on one interface, it forwards the frame on all of its other interfaces. In the next section, we will provide a more accurate explanation of how switches operate.

Now suppose that the host with IP address 222.222.222.220 wants to send an IP datagram to host 222.222.222.222. In this example, both the source and destination are in the same subnet, in the addressing sense of Section 4.3.3. To send a datagram, the source must give its adapter not only the IP datagram but also the MAC address for destination 222.222.222.222. The sending adapter will then construct a link-layer frame containing the destination's MAC address and send the frame into the LAN.

The important question addressed in this section is, How does the sending host determine the MAC address for the destination host with IP address 222.222.222.222? As you might have guessed, it uses ARP. An ARP module in the sending host takes any IP address on the same LAN as input, and returns the corresponding MAC address. In the example at hand, sending host 222.222.222.220 provides its ARP module the IP address 222.222.222.222, and the ARP module returns the corresponding MAC address 49-BD-D2-C7-56-2A.

So we see that ARP resolves an IP address to a MAC address. In many ways it is analogous to DNS (studied in Section 2.5), which resolves host names to IP addresses. However, one important difference between the two resolvers is that DNS resolves host names for hosts anywhere in the Internet, whereas ARP resolves IP addresses only for hosts and router interfaces on the same subnet. If a node in California were to try to use ARP to resolve the IP address for a node in Mississippi, ARP would return with an error.

IP Address	MAC Address	TTL
222.222.222.221	88-B2-2F-54-1A-0F	13:45:00
222.222.222.223	5C-66-AB-90-75-B1	13:52:00

Figure 6.18 ♦ A possible ARP table in 222.222.222.220

Now that we have explained what ARP does, let's look at how it works. Each host and router has an **ARP table** in its memory, which contains mappings of IP addresses to MAC addresses. Figure 6.18 shows what an ARP table in host 222.222.222.220 might look like. The ARP table also contains a time-to-live (TTL) value, which indicates when each mapping will be deleted from the table. Note that a table does not necessarily contain an entry for every host and router on the subnet; some may have never been entered into the table, and others may have expired. A typical expiration time for an entry is 20 minutes from when an entry is placed in an ARP table.

Now suppose that host 222.222.222.220 wants to send a datagram that is IP-addressed to another host or router on that subnet. The sending host needs to obtain the MAC address of the destination given the IP address. This task is easy if the sender's ARP table has an entry for the destination node. But what if the ARP table doesn't currently have an entry for the destination? In particular, suppose 222.222.222.220 wants to send a datagram to 222.222.222.222. In this case, the sender uses the ARP protocol to resolve the address. First, the sender constructs a special packet called an **ARP packet**. An ARP packet has several fields, including the sending and receiving IP and MAC addresses. Both ARP query and response packets have the same format. The purpose of the ARP query packet is to query all the other hosts and routers on the subnet to determine the MAC address corresponding to the IP address that is being resolved.

Returning to our example, 222.222.222.220 passes an ARP query packet to the adapter along with an indication that the adapter should send the packet to the MAC broadcast address, namely, FF-FF-FF-FF-FF-FF. The adapter encapsulates the ARP packet in a link-layer frame, uses the broadcast address for the frame's destination address, and transmits the frame into the subnet. Recalling our social security number/postal address analogy, an ARP query is equivalent to a person shouting out in a crowded room of cubicles in some company (say, AnyCorp): "What is the social security number of the person whose postal address is Cubicle 13, Room 112, AnyCorp, Palo Alto, California?" The frame containing the ARP query is received by all the other adapters on the subnet, and (because of the broadcast address) each adapter passes the ARP packet within the frame up to its ARP module. Each of these ARP modules checks to see if its IP address matches the destination IP address in the ARP packet. The one with a match sends back to the querying host a response ARP packet with the desired mapping. The querying host 222.222.222.220 can then update its ARP table and send its IP datagram, encapsulated in a link-layer frame whose destination MAC is that of the host or router responding to the earlier ARP query.

There are a couple of interesting things to note about the ARP protocol. First, the query ARP message is sent within a broadcast frame, whereas the response ARP message is sent within a standard frame. Before reading on you should think about why this is so. Second, ARP is plug-and-play; that is, an ARP table gets built automatically—it doesn't have to be configured by a system administrator. And if a host becomes disconnected from the subnet, its entry is eventually deleted from the other ARP tables in the subnet.

Students often wonder if ARP is a link-layer protocol or a network-layer protocol. As we've seen, an ARP packet is encapsulated within a link-layer frame and thus lies architecturally above the link layer. However, an ARP packet has fields containing link-layer addresses and thus is arguably a link-layer protocol, but it also contains network-layer addresses and thus is also arguably a network-layer protocol. In the end, ARP is probably best considered a protocol that straddles the boundary between the link and network layers—not fitting neatly into the simple layered protocol stack we studied in Chapter 1. Such are the complexities of real-world protocols!

Sending a Datagram off the Subnet

It should now be clear how ARP operates when a host wants to send a datagram to another host *on the same subnet.* But now let's look at the more complicated situation when a host on a subnet wants to send a network-layer datagram to a host *off the subnet* (that is, across a router onto another subnet). Let's discuss this issue in the context of Figure 6.19, which shows a simple network consisting of two subnets interconnected by a router.

There are several interesting things to note about Figure 6.19. Each host has exactly one IP address and one adapter. But, as discussed in Chapter 4, a router has an IP address for *each* of its interfaces. For each router interface there is also an ARP module (in the router) and an adapter. Because the router in Figure 6.19 has two interfaces, it has two IP addresses, two ARP modules, and two adapters. Of course, each adapter in the network has its own MAC address.

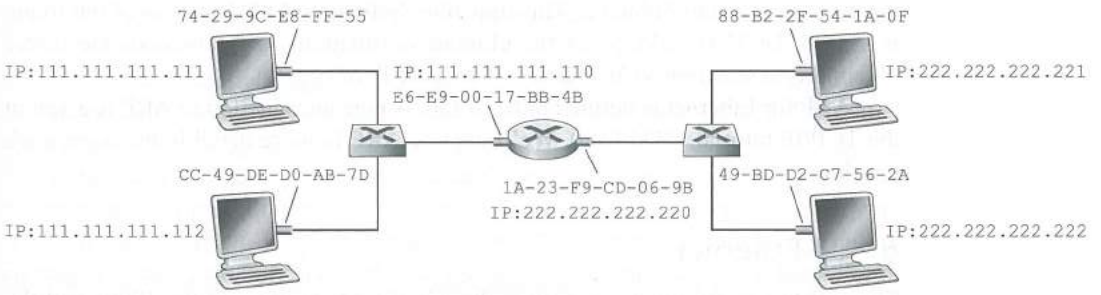


Figure 6.19 ♦ Two subnets interconnected by a router

Also note that Subnet 1 has the network address 111.111.111/24 and that Subnet 2 has the network address 222.222.222/24. Thus all of the interfaces connected to Subnet 1 have addresses of the form 111.111.111.xxx and all of the interfaces connected to Subnet 2 have addresses of the form 222.222.222.xxx.

Now let's examine how a host on Subnet 1 would send a datagram to a host on Subnet 2. Specifically, suppose that host 111.111.111.111 wants to send an IP datagram to a host 222.222.222.222. The sending host passes the datagram to its adapter, as usual. But the sending host must also indicate to its adapter an appropriate destination MAC address. What MAC address should the adapter use? One might be tempted to guess that the appropriate MAC address is that of the adapter for host 222.222.222.222, namely, 49-BD-D2-C7-56-2A. This guess, however, would be wrong! If the sending adapter were to use that MAC address, then none of the adapters on Subnet 1 would bother to pass the IP datagram up to its network layer, since the frame's destination address would not match the MAC address of any adapter on Subnet 1. The datagram would just die and go to datagram heaven.

If we look carefully at Figure 6.19, we see that in order for a datagram to go from 111.111.111.111 to a host on Subnet 2, the datagram must first be sent to the router interface 111.111.111.110, which is the IP address of the first-hop router on the path to the final destination. Thus, the appropriate MAC address for the frame is the address of the adapter for router interface 111.111.111.110, namely, E6-E9-00-17-BB-4B. How does the sending host acquire the MAC address for 111.111.111.110? By using ARP, of course! Once the sending adapter has this MAC address, it creates a frame (containing the datagram addressed to 222.222.222.222) and sends the frame into Subnet 1. The router adapter on Subnet 1 sees that the link-layer frame is addressed to it, and therefore passes the frame to the network layer of the router. Hooray—the IP datagram has successfully been moved from source host to the router! But we are not finished. We still have to move the datagram from the router to the destination. The router now has to determine the correct interface on which the datagram is to be forwarded. As discussed in Chapter 4, this is done by consulting a forwarding table in the router. The forwarding table tells the router that the datagram is to be forwarded via router interface 222.222.222.220. This interface then passes the datagram to its adapter, which encapsulates the datagram in a new frame and sends the frame into Subnet 2. This time, the destination MAC address of the frame is indeed the MAC address of the ultimate destination. And how does the router obtain this destination MAC address? From ARP, of course!

ARP for Ethernet is defined in RFC 826. A nice introduction to ARP is given in the TCP/IP tutorial, RFC 1180. We'll explore ARP in more detail in the homework problems.

6.4.2 Ethernet

Ethernet has pretty much taken over the wired LAN market. In the 1980s and the early 1990s, Ethernet faced many challenges from other LAN technologies, including

token ring, FDDI, and ATM. Some of these other technologies succeeded in capturing a part of the LAN market for a few years. But since its invention in the mid-1970s, Ethernet has continued to evolve and grow and has held on to its dominant position. Today, Ethernet is by far the most prevalent wired LAN technology, and it is likely to remain so for the foreseeable future. One might say that Ethernet has been to local area networking what the Internet has been to global networking.

There are many reasons for Ethernet's success. First, Ethernet was the first widely deployed high-speed LAN. Because it was deployed early, network administrators became intimately familiar with Ethernet—its wonders and its quirks—and were reluctant to switch over to other LAN technologies when they came on the scene. Second, token ring, FDDI, and ATM were more complex and expensive than Ethernet, which further discouraged network administrators from switching over. Third, the most compelling reason to switch to another LAN technology (such as FDDI or ATM) was usually the higher data rate of the new technology; however, Ethernet always fought back, producing versions that operated at equal data rates or higher. Switched Ethernet was also introduced in the early 1990s, which further increased its effective data rates. Finally, because Ethernet has been so popular, Ethernet hardware (in particular, adapters and switches) has become a commodity and is remarkably cheap.

The original Ethernet LAN was invented in the mid-1970s by Bob Metcalfe and David Boggs. The original Ethernet LAN used a coaxial bus to interconnect the nodes. Bus topologies for Ethernet actually persisted throughout the 1980s and into the mid-1990s. Ethernet with a bus topology is a broadcast LAN—all transmitted frames travel to and are processed by *all* adapters connected to the bus. Recall that we covered Ethernet's CSMA/CD multiple access protocol with binary exponential backoff in Section 6.3.2.

By the late 1990s, most companies and universities had replaced their LANs with Ethernet installations using a hub-based star topology. In such an installation the hosts (and routers) are directly connected to a hub with twisted-pair copper wire. A **hub** is a physical-layer device that acts on individual bits rather than frames. When a bit, representing a zero or a one, arrives from one interface, the hub simply re-creates the bit, boosts its energy strength, and transmits the bit onto all the other interfaces. Thus, Ethernet with a hub-based star topology is also a broadcast LAN—whenever a hub receives a bit from one of its interfaces, it sends a copy out on all of its other interfaces. In particular, if a hub receives frames from two different interfaces at the same time, a collision occurs and the nodes that created the frames must retransmit.

In the early 2000s Ethernet experienced yet another major evolutionary change. Ethernet installations continued to use a star topology, but the hub at the center was replaced with a **switch**. We'll be examining switched Ethernet in depth later in this chapter. For now, we only mention that a switch is not only "collision-less" but is also a bona-fide store-and-forward packet switch; but unlike routers, which operate up through layer 3, a switch operates only up through layer 2.

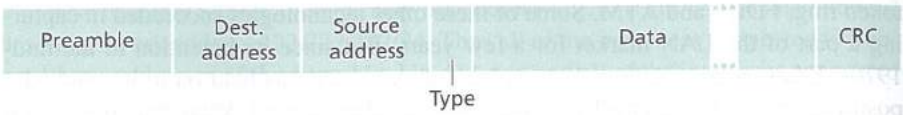


Figure 6.20 ♦ Ethernet frame structure

Ethernet Frame Structure

We can learn a lot about Ethernet by examining the Ethernet frame, which is shown in Figure 6.20. To give this discussion about Ethernet frames a tangible context, let's consider sending an IP datagram from one host to another host, with both hosts on the same Ethernet LAN (for example, the Ethernet LAN in Figure 6.17.) (Although the payload of our Ethernet frame is an IP datagram, we note that an Ethernet frame can carry other network-layer packets as well.) Let the sending adapter, adapter A, have the MAC address AA-AA-AA-AA-AA-AA and the receiving adapter, adapter B, have the MAC address BB-BB-BB-BB-BB-BB. The sending adapter encapsulates the IP datagram within an Ethernet frame and passes the frame to the physical layer. The receiving adapter receives the frame from the physical layer, extracts the IP datagram, and passes the IP datagram to the network layer. In this context, let's now examine the six fields of the Ethernet frame, as shown in Figure 6.20.

- *Data field (46 to 1,500 bytes).* This field carries the IP datagram. The maximum transmission unit (MTU) of Ethernet is 1,500 bytes. This means that if the IP datagram exceeds 1,500 bytes, then the host has to fragment the datagram, as discussed in Section 4.3.2. The minimum size of the data field is 46 bytes. This means that if the IP datagram is less than 46 bytes, the data field has to be "stuffed" to fill it out to 46 bytes. When stuffing is used, the data passed to the network layer contains the stuffing as well as an IP datagram. The network layer uses the length field in the IP datagram header to remove the stuffing.
- *Destination address (6 bytes).* This field contains the MAC address of the destination adapter, BB-BB-BB-BB-BB-BB. When adapter B receives an Ethernet frame whose destination address is either BB-BB-BB-BB-BB-BB or the MAC broadcast address, it passes the contents of the frame's data field to the network layer; if it receives a frame with any other MAC address, it discards the frame.
- *Source address (6 bytes).* This field contains the MAC address of the adapter that transmits the frame onto the LAN, in this example, AA-AA-AA-AA-AA-AA.
- *Type field (2 bytes).* The type field permits Ethernet to multiplex network-layer protocols. To understand this, we need to keep in mind that hosts can use other network-layer protocols besides IP. In fact, a given host may support multiple network-layer protocols using different protocols for different applications.

For this reason, when the Ethernet frame arrives at adapter B, adapter B needs to know to which network-layer protocol it should pass (that is, demultiplex) the contents of the data field. IP and other network-layer protocols (for example, Novell IPX or AppleTalk) each have their own, standardized type number. Furthermore, the ARP protocol (discussed in the previous section) has its own type number, and if the arriving frame contains an ARP packet (i.e., has a type field of 0806 hexadecimal), the ARP packet will be demultiplexed up to the ARP protocol. Note that the type field is analogous to the protocol field in the network-layer datagram and the port-number fields in the transport-layer segment; all of these fields serve to glue a protocol at one layer to a protocol at the layer above.

- *Cyclic redundancy check (CRC) (4 bytes).* As discussed in Section 6.2.3, the purpose of the CRC field is to allow the receiving adapter, adapter B, to detect bit errors in the frame.
- *Preamble (8 bytes).* The Ethernet frame begins with an 8-byte preamble field. Each of the first 7 bytes of the preamble has a value of 10101010; the last byte is 10101011. The first 7 bytes of the preamble serve to "wake up" the receiving adapters and to synchronize their clocks to that of the sender's clock. Why should the clocks be out of synchronization? Keep in mind that adapter A aims to transmit the frame at 10 Mbps, 100 Mbps, or 1 Gbps, depending on the type of Ethernet LAN. However, because nothing is absolutely perfect, adapter A will not transmit the frame at exactly the target rate; there will always be some *drift* from the target rate, a drift which is not known *a priori* by the other adapters on the LAN. A receiving adapter can lock onto adapter A's clock simply by locking onto the bits in the first 7 bytes of the preamble. The last 2 bits of the eighth byte of the preamble (the first two consecutive 1s) alert adapter B that the "important stuff" is about to come.

All of the Ethernet technologies provide connectionless service to the network layer. That is, when adapter A wants to send a datagram to adapter B, adapter A encapsulates the datagram in an Ethernet frame and sends the frame into the LAN, without first handshaking with adapter B. This layer-2 connectionless service is analogous to IP's layer-3 datagram service and UDP's layer-4 connectionless service.

Ethernet technologies provide an unreliable service to the network layer. Specifically, when adapter B receives a frame from adapter A, it runs the frame through a CRC check, but neither sends an acknowledgment when a frame passes the CRC check nor sends a negative acknowledgment when a frame fails the CRC check. When a frame fails the CRC check, adapter B simply discards the frame. Thus, adapter A has no idea whether its transmitted frame reached adapter B and passed the CRC check. This lack of reliable transport (at the link layer) helps to make Ethernet simple and cheap. But it also means that the stream of datagrams passed to the network layer can have gaps.

puters, which in n
1980s. Metcalfe
ner. So armed wi
protocols, Metcal

Metcalfe and
hosts separated b
the researchers a
then forged an al
10 Mbps Etherne
commercializing
developed and c
ogy. In particular
for the immensely

RY

orked
amson's
nd just
e vis-
etting a
Alto com-
of the
e man-
access
to 256
g most of
. Metcalfe
et as a
interest in
om, which
echnol-
ly 1980s

If there are g
Host B see gaps
the application is
application in Ho
application is us
contained in disc
when TCP retrans
at which it was
although Etherne
with brand-new d
mitted at least on

Ethernet Techn

In our discussion
standard. But in fa
dering acronyms

plication at
on whether
P, then the
hand, if the
ge the data
t. Note that
rnet adapter
nsmit data,
w datagram
been trans-

gle protocol
ewhat bewil-
0BASE-LX,

10GBASE-T and 40GBASE-T. These and many other Ethernet technologies have been standardized over the years by the IEEE 802.3 CSMA/CD (Ethernet) working group [IEEE 802.3 2012]. While these acronyms may appear bewildering, there is actually considerable order here. The first part of the acronym refers to the speed of the standard: 10, 100, 1000, or 10G, for 10 Megabit (per second), 100 Megabit, Gigabit, 10 Gigabit and 40 Gigibit Ethernet, respectively. "BASE" refers to baseband Ethernet, meaning that the physical media only carries Ethernet traffic; almost all of the 802.3 standards are for baseband Ethernet. The final part of the acronym refers to the physical media itself; Ethernet is both a link-layer *and* a physical-layer specification and is carried over a variety of physical media including coaxial cable, copper wire, and fiber. Generally, a "T" refers to twisted-pair copper wires.

Historically, an Ethernet was initially conceived of as a segment of coaxial cable. The early 10BASE-2 and 10BASE-5 standards specify 10 Mbps Ethernet over two types of coaxial cable, each limited in length to 500 meters. Longer runs could be obtained by using a **repeater**—a physical-layer device that receives a signal on the input side, and regenerates the signal on the output side. A coaxial cable corresponds nicely to our view of Ethernet as a broadcast medium—all frames transmitted by one interface are received at other interfaces, and Ethernet's CDMA/CD protocol nicely solves the multiple access problem. Nodes simply attach to the cable, and *voila*, we have a local area network!

Ethernet has passed through a series of evolutionary steps over the years, and today's Ethernet is very different from the original bus-topology designs using coaxial cable. In most installations today, nodes are connected to a switch via point-to-point segments made of twisted-pair copper wires or fiber-optic cables, as shown in Figures 6.15–6.17.

In the mid-1990s, Ethernet was standardized at 100 Mbps, 10 times faster than 10 Mbps Ethernet. The original Ethernet MAC protocol and frame format were preserved, but higher-speed physical layers were defined for copper wire (100BASE-T) and fiber (100BASE-FX, 100BASE-SX, 100BASE-BX). Figure 6.21 shows these different standards and the common Ethernet MAC protocol and frame format. 100 Mbps Ethernet is limited to a 100-meter distance over twisted pair, and to

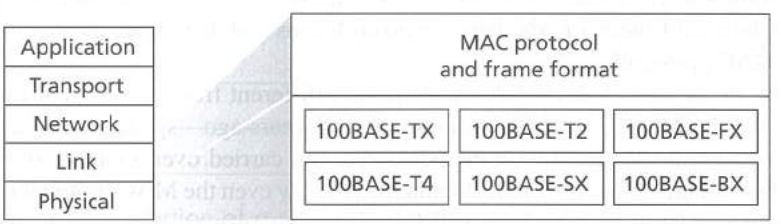


Figure 6.21 ♦ 100 Mbps Ethernet standards: A common link layer, different physical layers

several kilometers over fiber, allowing Ethernet switches in different buildings to be connected.

Gigabit Ethernet is an extension to the highly successful 10 Mbps and 100 Mbps Ethernet standards. Offering a raw data rate of 40,000 Mbps, 40 Gigabit Ethernet maintains full compatibility with the huge installed base of Ethernet equipment. The standard for Gigabit Ethernet, referred to as IEEE 802.3z, does the following:

- Uses the standard Ethernet frame format (Figure 6.20) and is backward compatible with 10BASE-T and 100BASE-T technologies. This allows for easy integration of Gigabit Ethernet with the existing installed base of Ethernet equipment.
- Allows for point-to-point links as well as shared broadcast channels. Point-to-point links use switches while broadcast channels use hubs, as described earlier. In Gigabit Ethernet jargon, hubs are called *buffered distributors*.
- Uses CSMA/CD for shared broadcast channels. In order to have acceptable efficiency, the maximum distance between nodes must be severely restricted.
- Allows for full-duplex operation at 40 Gbps in both directions for point-to-point channels.

Initially operating over optical fiber, Gigabit Ethernet is now able to run over category 5 UTP cabling.

Let's conclude our discussion of Ethernet technology by posing a question that may have begun troubling you. In the days of bus topologies and hub-based star topologies, Ethernet was clearly a broadcast link (as defined in Section 6.3) in which frame collisions occurred when nodes transmitted at the same time. To deal with these collisions, the Ethernet standard included the CSMA/CD protocol, which is particularly effective for a wired broadcast LAN spanning a small geographical region. But if the prevalent use of Ethernet today is a switch-based star topology, using store-and-forward packet switching, is there really a need anymore for an Ethernet MAC protocol? As we'll see shortly, a switch coordinates its transmissions and never forwards more than one frame onto the same interface at any time. Furthermore, modern switches are full-duplex, so that a switch and a node can each send frames to each other at the same time without interference. In other words, in a switch-based Ethernet LAN there are no collisions and, therefore, there is no need for a MAC protocol!

As we've seen, today's Ethernets are *very* different from the original Ethernet conceived by Metcalfe and Boggs more than 30 years ago—speeds have increased by three orders of magnitude, Ethernet frames are carried over a variety of media, switched-Ethernets have become dominant, and now even the MAC protocol is often unnecessary! Is all of this *really* still Ethernet? The answer, of course, is "yes, by definition." It is interesting to note, however, that through all of these changes, there

has indeed been one enduring constant that has remained unchanged over 30 years—Ethernet's frame format. Perhaps this then is the one true and timeless centerpiece of the Ethernet standard.

6.4.3 Link-Layer Switches

Up until this point, we have been purposefully vague about what a switch actually does and how it works. The role of the switch is to receive incoming link-layer frames and forward them onto outgoing links; we'll study this forwarding function in detail in this subsection. We'll see that the switch itself is **transparent** to the hosts and routers in the subnet; that is, a host/router addresses a frame to another host/router (rather than addressing the frame to the switch) and happily sends the frame into the LAN, unaware that a switch will be receiving the frame and forwarding it. The rate at which frames arrive to any one of the switch's output interfaces may temporarily exceed the link capacity of that interface. To accommodate this problem, switch output interfaces have buffers, in much the same way that router output interfaces have buffers for datagrams. Let's now take a closer look at how switches operate.

Forwarding and Filtering

Filtering is the switch function that determines whether a frame should be forwarded to some interface or should just be dropped. **Forwarding** is the switch function that determines the interfaces to which a frame should be directed, and then moves the frame to those interfaces. Switch filtering and forwarding are done with a **switch table**. The switch table contains entries for some, but not necessarily all, of the hosts and routers on a LAN. An entry in the switch table contains (1) a MAC address, (2) the switch interface that leads toward that MAC address, and (3) the time at which the entry was placed in the table. An example switch table for the uppermost switch in Figure 6.15 is shown in Figure 6.22. This description of frame forwarding may sound similar to our discussion of datagram forwarding

Address	Interface	Time
62-FE-F7-11-89-A3	1	9:32
7C-BA-B2-B4-91-10	3	9:36
....	....	....

Figure 6.22 ♦ Portion of a switch table for the uppermost switch in Figure 6.15

in Chapter 4. Indeed, in our discussion of generalized forwarding in Section 4.4, we learned that many modern packet switches can be configured to forward on the basis of layer-2 destination MAC addresses (i.e., function as a layer-2 switch) or layer-3 IP destination addresses (i.e., function as a layer-3 router). Nonetheless, we'll make the important distinction that switches forward packets based on MAC addresses rather than on IP addresses. We will also see that a traditional (i.e., in a non-SDN context) switch table is constructed in a very different manner from a router's forwarding table.

To understand how switch filtering and forwarding work, suppose a frame with destination address DD-DD-DD-DD-DD-DD arrives at the switch on interface x. The switch indexes its table with the MAC address DD-DD-DD-DD-DD-DD. There are three possible cases:

- There is no entry in the table for DD-DD-DD-DD-DD-DD. In this case, the switch forwards copies of the frame to the output buffers preceding *all* interfaces except for interface x. In other words, if there is no entry for the destination address, the switch broadcasts the frame.
- There is an entry in the table, associating DD-DD-DD-DD-DD-DD with interface x. In this case, the frame is coming from a LAN segment that contains adapter DD-DD-DD-DD-DD-DD. There being no need to forward the frame to any of the other interfaces, the switch performs the filtering function by discarding the frame.
- There is an entry in the table, associating DD-DD-DD-DD-DD-DD with interface $y \neq x$. In this case, the frame needs to be forwarded to the LAN segment attached to interface y. The switch performs its forwarding function by putting the frame in an output buffer that precedes interface y.

Let's walk through these rules for the uppermost switch in Figure 6.15 and its switch table in Figure 6.22. Suppose that a frame with destination address 62-FE-F7-11-89-A3 arrives at the switch from interface 1. The switch examines its table and sees that the destination is on the LAN segment connected to interface 1 (that is, Electrical Engineering). This means that the frame has already been broadcast on the LAN segment that contains the destination. The switch therefore filters (that is, discards) the frame. Now suppose a frame with the same destination address arrives from interface 2. The switch again examines its table and sees that the destination is in the direction of interface 1; it therefore forwards the frame to the output buffer preceding interface 1. It should be clear from this example that as long as the switch table is complete and accurate, the switch forwards frames toward destinations without any broadcasting.

In this sense, a switch is "smarter" than a hub. But how does this switch table get configured in the first place? Are there link-layer equivalents to network-layer routing protocols? Or must an overworked manager manually configure the switch table?

Self-Learning

A switch has the wonderful property (particularly for the already-overworked network administrator) that its table is built automatically, dynamically, and autonomously—without any intervention from a network administrator or from a configuration protocol. In other words, switches are **self-learning**. This capability is accomplished as follows:

1. The switch table is initially empty.
2. For each incoming frame received on an interface, the switch stores in its table (1) the MAC address in the frame's *source address field*, (2) the interface from which the frame arrived, and (3) the current time. In this manner the switch records in its table the LAN segment on which the sender resides. If every host in the LAN eventually sends a frame, then every host will eventually get recorded in the table.
3. The switch deletes an address in the table if no frames are received with that address as the source address after some period of time (the **aging time**). In this manner, if a PC is replaced by another PC (with a different adapter), the MAC address of the original PC will eventually be purged from the switch table.

Let's walk through the self-learning property for the uppermost switch in Figure 6.15 and its corresponding switch table in Figure 6.22. Suppose at time 9:39 a frame with source address 01-12-23-34-45-56 arrives from interface 2. Suppose that this address is not in the switch table. Then the switch adds a new entry to the table, as shown in Figure 6.23.

Continuing with this same example, suppose that the aging time for this switch is 60 minutes, and no frames with source address 62-FE-F7-11-89-A3 arrive to the switch between 9:32 and 10:32. Then at time 10:32, the switch removes this address from its table.

Address	Interface	Time
01-12-23-34-45-56	2	9:39
62-FE-F7-11-89-A3	1	9:32
7C-BA-B2-B4-91-10	3	9:36
....	....	....

Figure 6.23 ♦ Switch learns about the location of an adapter with address 01-12-23-34-45-56

Switches are **plug-and-play devices** because they require no intervention from a network administrator or user. A network administrator wanting to install a switch need do nothing more than connect the LAN segments to the switch interfaces. The administrator need not configure the switch tables at the time of installation or when a host is removed from one of the LAN segments. Switches are also full-duplex, meaning any switch interface can send and receive at the same time.

Properties of Link-Layer Switching

Having described the basic operation of a link-layer switch, let's now consider their features and properties. We can identify several advantages of using switches, rather than broadcast links such as buses or hub-based star topologies:

- *Elimination of collisions.* In a LAN built from switches (and without hubs), there is no wasted bandwidth due to collisions! The switches buffer frames and never transmit more than one frame on a segment at any one time. As with a router, the maximum aggregate throughput of a switch is the sum of all the switch interface rates. Thus, switches provide a significant performance improvement over LANs with broadcast links.
- *Heterogeneous links.* Because a switch isolates one link from another, the different links in the LAN can operate at different speeds and can run over different media. For example, the uppermost switch in Figure 6.15 might have three1 Gbps 1000BASE-T copper links, two 100 Mbps 100BASE-FX fiber links, and one 100BASE-T copper link. Thus, a switch is ideal for mixing legacy equipment with new equipment.
- *Management.* In addition to providing enhanced security (see sidebar on Focus on Security), a switch also eases network management. For example, if an adapter malfunctions and continually sends Ethernet frames (called a jabbering adapter), a switch can detect the problem and internally disconnect the malfunctioning adapter. With this feature, the network administrator need not get out of bed and drive back to work in order to correct the problem. Similarly, a cable cut disconnects only that host that was using the cut cable to connect to the switch. In the days of coaxial cable, many a network manager spent hours "walking the line" (or more accurately, "crawling the floor") to find the cable break that brought down the entire network. Switches also gather statistics on bandwidth usage, collision rates, and traffic types, and make this information available to the network manager. This information can be used to debug and correct problems, and to plan how the LAN should evolve in the future. Researchers are exploring adding yet more management functionality into Ethernet LANs in prototype deployments [Casado 2007; Koponen 2011].

FOCUS ON SECURITY

SNIFFING A SWITCHED LAN: SWITCH POISONING

When a host is connected to a switch, it typically only receives frames that are intended for it. For example, consider a switched LAN in Figure 6.17. When host A sends a frame to host B, and there is an entry for host B in the switch table, then the switch will forward the frame *only* to host B. If host C happens to be running a sniffer, host C will not be able to sniff this A-to-B frame. Thus, in a switched-LAN environment (in contrast to a broadcast link environment such as 802.11 LANs or hub–based Ethernet LANs), it is more difficult for an attacker to sniff frames. *However*, because the switch broadcasts frames that have destination addresses that are not in the switch table, the sniffer at C can still sniff some frames that are not intended for C. Furthermore, a sniffer will be able sniff all Ethernet broadcast frames with broadcast destination address FF–FF–FF–FF–FF–FF. A well-known attack against a switch, called **switch poisoning**, is to send tons of packets to the switch with many different bogus source MAC addresses, thereby filling the switch table with bogus entries and leaving no room for the MAC addresses of the legitimate hosts. This causes the switch to broadcast most frames, which can then be picked up by the sniffer [Skoudis 2006]. As this attack is rather involved even for a sophisticated attacker, switches are significantly less vulnerable to sniffing than are hubs and wireless LANs.

Switches Versus Routers

As we learned in Chapter 4, routers are store-and-forward packet switches that forward packets using network-layer addresses. Although a switch is also a store-and-forward packet switch, it is fundamentally different from a router in that it forwards packets using MAC addresses. Whereas a router is a layer-3 packet switch, a switch is a layer-2 packet switch. Recall, however, that we learned in Section 4.4 that modern switches using the "match plus action" operation can be used to forward a layer-2 frame based on the frame's destination MAC address, as well as a layer-3 datagram using the datagram's destination IP address. Indeed, we saw that switches using the OpenFlow standard can perform generalized packet forwarding based on any of eleven different frame, datagram, and transport-layer header fields.

Even though switches and routers are fundamentally different, network administrators must often choose between them when installing an interconnection device. For example, for the network in Figure 6.15, the network administrator could just as easily have used a router instead of a switch to connect the department LANs, servers, and internet gateway router. Indeed, a router would permit interdepartmental communication without creating collisions. Given that both switches and routers are candidates for interconnection devices, what are the pros and cons of the two approaches?

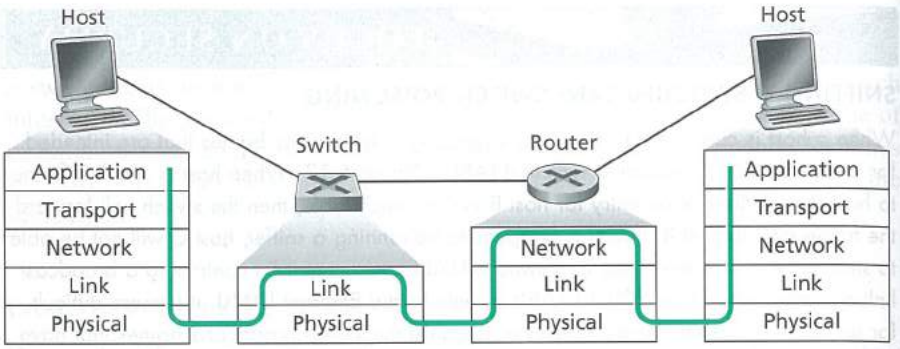


Figure 6.24 ♦ Packet processing in switches, routers, and hosts

First consider the pros and cons of switches. As mentioned above, switches are plug-and-play, a property that is cherished by all the overworked network administrators of the world. Switches can also have relatively high filtering and forwarding rates—as shown in Figure 6.24, switches have to process frames only up through layer 2, whereas routers have to process datagrams up through layer 3. On the other hand, to prevent the cycling of broadcast frames, the active topology of a switched network is restricted to a spanning tree. Also, a large switched network would require large ARP tables in the hosts and routers and would generate substantial ARP traffic and processing. Furthermore, switches are susceptible to broadcast storms—if one host goes haywire and transmits an endless stream of Ethernet broadcast frames, the switches will forward all of these frames, causing the entire network to collapse.

Now consider the pros and cons of routers. Because network addressing is often hierarchical (and not flat, as is MAC addressing), packets do not normally cycle through routers even when the network has redundant paths. (However, packets can cycle when router tables are misconfigured; but as we learned in Chapter 4, IP uses a special datagram header field to limit the cycling.) Thus, packets are not restricted to a spanning tree and can use the best path between source and destination. Because routers do not have the spanning tree restriction, they have allowed the Internet to be built with a rich topology that includes, for example, multiple active links between Europe and North America. Another feature of routers is that they provide firewall protection against layer-2 broadcast storms. Perhaps the most significant drawback of routers, though, is that they are not plug-and-play—they and the hosts that connect to them need their IP addresses to be configured. Also, routers often have a larger per-packet processing time than switches, because they have to process up through the layer-3 fields. Finally, there are two different ways to pronounce the word *router*, either as "rootor" or as "rowter," and people waste a lot of time arguing over the proper pronunciation [Perlman 1999].

Given that both switches and routers have their pros and cons (as summarized in Table 6.1), when should an institutional network (for example, a university campus

	Hubs	Routers	Switches
Traffic isolation	No	Yes	Yes
Plug and play	Yes	No	Yes
Optimal routing	No	Yes	No

Table 6.1 ♦ Comparison of the typical features of popular interconnection devices

network or a corporate campus network) use switches, and when should it use routers? Typically, small networks consisting of a few hundred hosts have a few LAN segments. Switches suffice for these small networks, as they localize traffic and increase aggregate throughput without requiring any configuration of IP addresses. But larger networks consisting of thousands of hosts typically include routers within the network (in addition to switches). The routers provide a more robust isolation of traffic, control broadcast storms, and use more "intelligent" routes among the hosts in the network.

For more discussion of the pros and cons of switched versus routed networks, as well as a discussion of how switched LAN technology can be extended to accommodate two orders of magnitude more hosts than today's Ethernets, see [Meyers 2004; Kim 2008].

6.4.4 Virtual Local Area Networks (VLANs)

In our earlier discussion of Figure 6.15, we noted that modern institutional LANs are often configured hierarchically, with each workgroup (department) having its own switched LAN connected to the switched LANs of other groups via a switch hierarchy. While such a configuration works well in an ideal world, the real world is often far from ideal. Three drawbacks can be identified in the configuration in Figure 6.15:

- *Lack of traffic isolation.* Although the hierarchy localizes group traffic to within a single switch, broadcast traffic (e.g., frames carrying ARP and DHCP messages or frames whose destination has not yet been learned by a self-learning switch) must still traverse the entire institutional network. Limiting the scope of such broadcast traffic would improve LAN performance. Perhaps more importantly, it also may be desirable to limit LAN broadcast traffic for security/privacy reasons. For example, if one group contains the company's executive management team and another group contains disgruntled employees running Wireshark packet sniffers, the network manager may well prefer that the executives' traffic never even reaches employee hosts. This type of isolation could be provided by

replacing the center switch in Figure 6.15 with a router. We'll see shortly that this isolation also can be achieved via a switched (layer 2) solution.

- *Inefficient use of switches.* If instead of three groups, the institution had 10 groups, then 10 first-level switches would be required. If each group were small, say less than 10 people, then a single 96-port switch would likely be large enough to accommodate everyone, but this single switch would not provide traffic isolation.
- *Managing users.* If an employee moves between groups, the physical cabling must be changed to connect the employee to a different switch in Figure 6.15. Employees belonging to two groups make the problem even harder.

Fortunately, each of these difficulties can be handled by a switch that supports **virtual local area networks (VLANs)**. As the name suggests, a switch that supports VLANs allows multiple *virtual* local area networks to be defined over a single *physical* local area network infrastructure. Hosts within a VLAN communicate with each other as if they (and no other hosts) were connected to the switch. In a port-based VLAN, the switch's ports (interfaces) are divided into groups by the network manager. Each group constitutes a VLAN, with the ports in each VLAN forming a broadcast domain (i.e., broadcast traffic from one port can only reach other ports in the group). Figure 6.25 shows a single switch with 16 ports. Ports 2 to 8 belong to the EE VLAN, while ports 9 to 15 belong to the CS VLAN (ports 1 and 16 are unassigned). This VLAN solves all of the difficulties noted above—EE and CS VLAN frames are isolated from each other, the two switches in Figure 6.15 have been replaced by a single switch, and if the user at switch port 8 joins the CS Department, the network operator simply reconfigures the VLAN software so that port 8 is now associated with the CS VLAN. One can easily imagine how the VLAN switch is configured and operates—the network manager declares a port to belong

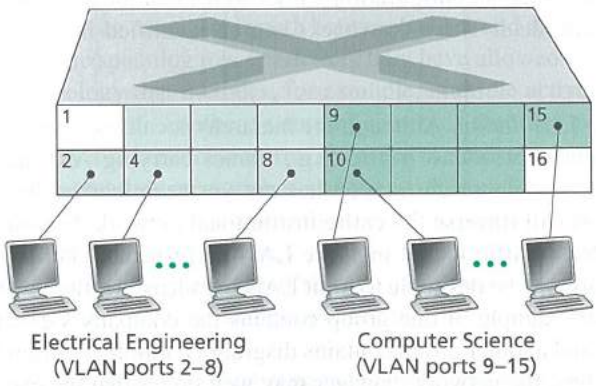


Figure 6.25 ♦ A single switch with two configured VLANs

to a given VLAN (with undeclared ports belonging to a default VLAN) using switch management software, a table of port-to-VLAN mappings is maintained within the switch; and switch hardware only delivers frames between ports belonging to the same VLAN.

But by completely isolating the two VLANs, we have introduced a new difficulty! How can traffic from the EE Department be sent to the CS Department? One way to handle this would be to connect a VLAN switch port (e.g., port 1 in Figure 6.25) to an external router and configure that port to belong both the EE and CS VLANs. In this case, even though the EE and CS departments share the same physical switch, the logical configuration would look as if the EE and CS departments had separate switches connected via a router. An IP datagram going from the EE to the CS department would first cross the EE VLAN to reach the router and then be forwarded by the router back over the CS VLAN to the CS host. Fortunately, switch vendors make such configurations easy for the network manager by building a single device that contains both a VLAN switch *and* a router, so a separate external router is not needed. A homework problem at the end of the chapter explores this scenario in more detail.

Returning again to Figure 6.15, let's now suppose that rather than having a separate Computer Engineering department, some EE and CS faculty are housed in a separate building, where (of course!) they need network access, and (of course!) they'd like to be part of their department's VLAN. Figure 6.26 shows a second 8-port switch, where the switch ports have been defined as belonging to the EE or the CS VLAN, as needed. But how should these two switches be interconnected? One easy solution would be to define a port belonging to the CS VLAN on each switch (similarly for the EE VLAN) and to connect these ports to each other, as shown in Figure 6.26(a). This solution doesn't scale, however, since N VLANS would require N ports on each switch simply to interconnect the two switches.

A more scalable approach to interconnecting VLAN switches is known as **VLAN trunking**. In the VLAN trunking approach shown in Figure 6.26(b), a special port on each switch (port 16 on the left switch and port 1 on the right switch) is configured as a trunk port to interconnect the two VLAN switches. The trunk port belongs to all VLANs, and frames sent to any VLAN are forwarded over the trunk link to the other switch. But this raises yet another question: How does a switch know that a frame arriving on a trunk port belongs to a particular VLAN? The IEEE has defined an extended Ethernet frame format, 802.1Q, for frames crossing a VLAN trunk. As shown in Figure 6.27, the 802.1Q frame consists of the standard Ethernet frame with a four-byte **VLAN tag** added into the header that carries the identity of the VLAN to which the frame belongs. The VLAN tag is added into a frame by the switch at the sending side of a VLAN trunk, parsed, and removed by the switch at the receiving side of the trunk. The VLAN tag itself consists of a 2-byte Tag Protocol Identifier (TPID) field (with a fixed hexadecimal value of 81-00), a 2-byte Tag Control Information field that contains a 12-bit VLAN identifier field, and a 3-bit priority field that is similar in intent to the IP datagram TOS field.

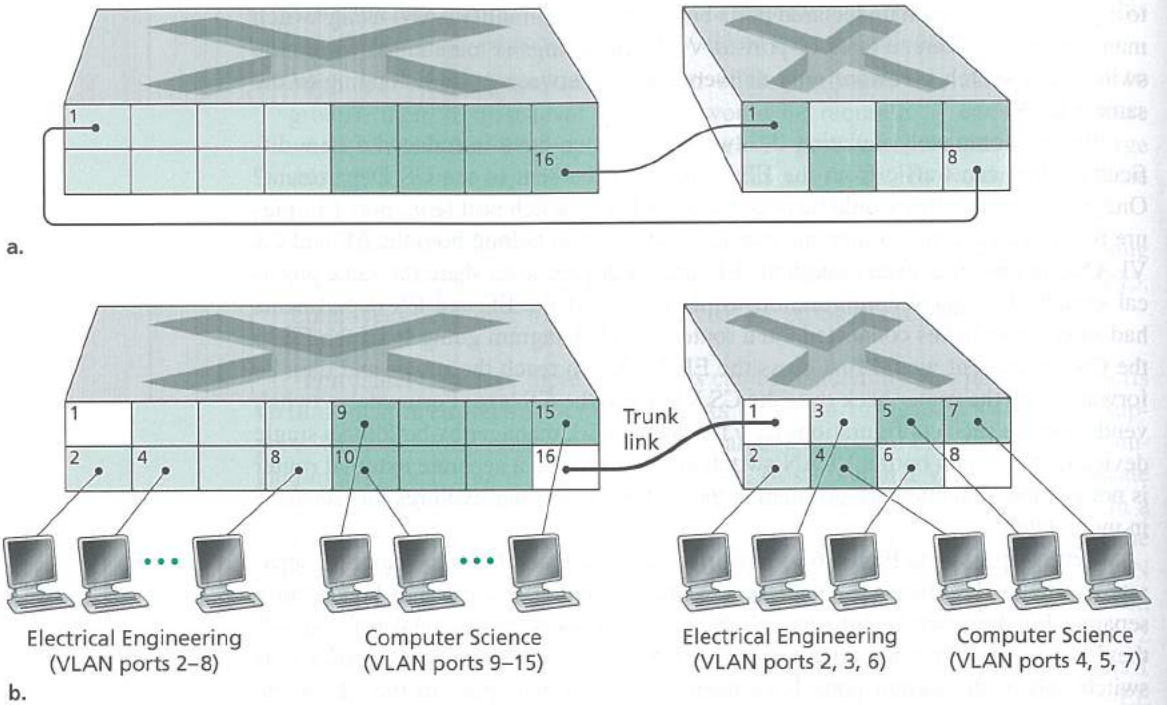


Figure 6.26 ♦ Connecting two VLAN switches with two VLANs: (a) two cables (b) trunked

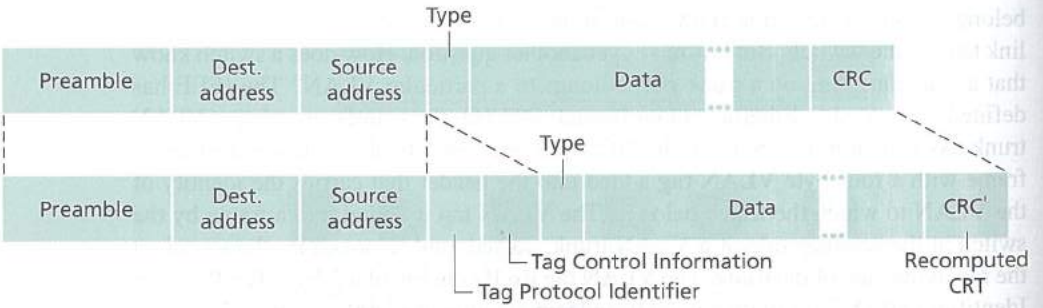


Figure 6.27 ♦ Original Ethernet frame (top), 802.1Q-tagged Ethernet VLAN frame (below)

In this discussion, we've only briefly touched on VLANs and have focused on port-based VLANs. We should also mention that VLANs can be defined in several other ways. In MAC-based VLANs, the network manager specifies the set of MAC addresses that belong to each VLAN; whenever a device attaches to a port, the port is connected into the appropriate VLAN based on the MAC address of the device. VLANs can also be defined based on network-layer protocols (e.g., IPv4, IPv6, or Appletalk) and other criteria. It is also possible for VLANs to be extended across IP routers, allowing islands of LANs to be connected together to form a single VLAN that could span the globe [Yu 2011]. See the 802.1Q standard [IEEE 802.1q 2005] for more details.

6.5 Link Virtualization: A Network as a Link Layer

Because this chapter concerns link-layer protocols, and given that we're now nearing the chapter's end, let's reflect on how our understanding of the term *link* has evolved. We began this chapter by viewing the link as a physical wire connecting two communicating hosts. In studying multiple access protocols, we saw that multiple hosts could be connected by a shared wire and that the "wire" connecting the hosts could be radio spectra or other media. This led us to consider the link a bit more abstractly as a channel, rather than as a wire. In our study of Ethernet LANs (Figure 6.15) we saw that the interconnecting media could actually be a rather complex switched infrastructure. Throughout this evolution, however, the hosts themselves maintained the view that the interconnecting medium was simply a link-layer channel connecting two or more hosts. We saw, for example, that an Ethernet host can be blissfully unaware of whether it is connected to other LAN hosts by a single short LAN segment (Figure 6.17) or by a geographically dispersed switched LAN (Figure 6.15) or by a VLAN (Figure 6.26).

In the case of a dialup modem connection between two hosts, the link connecting the two hosts is actually the telephone network—a logically separate, global telecommunications network with its own switches, links, and protocol stacks for data transfer and signaling. From the Internet link-layer point of view, however, the dial-up connection through the telephone network is viewed as a simple "wire." In this sense, the Internet virtualizes the telephone network, viewing the telephone network as a link-layer technology providing link-layer connectivity between two Internet hosts. You may recall from our discussion of overlay networks in Chapter 2 that an overlay network similarly views the Internet as a means for providing connectivity between overlay nodes, seeking to overlay the Internet in the same way that the Internet overlays the telephone network.

In this section, we'll consider Multiprotocol Label Switching (MPLS) networks. Unlike the circuit-switched telephone network, MPLS is a packet-switched,

virtual-circuit network in its own right. It has its own packet formats and forwarding behaviors. Thus, from a pedagogical viewpoint, a discussion of MPLS fits well into a study of either the network layer or the link layer. From an Internet viewpoint, however, we can consider MPLS, like the telephone network and switched-Ethernets, as a link-layer technology that serves to interconnect IP devices. Thus, we'll consider MPLS in our discussion of the link layer. Frame-relay and ATM networks can also be used to interconnect IP devices, though they represent a slightly older (but still deployed) technology and will not be covered here; see the very readable book [Goralski 1999] for details. Our treatment of MPLS will be necessarily brief, as entire books could be (and have been) written on these networks. We recommend [Davie 2000] for details on MPLS. We'll focus here primarily on how MPLS servers interconnect to IP devices, although we'll dive a bit deeper into the underlying technologies as well.

6.5.1 Multiprotocol Label Switching (MPLS)

Multiprotocol Label Switching (MPLS) evolved from a number of industry efforts in the mid-to-late 1990s to improve the forwarding speed of IP routers by adopting a key concept from the world of virtual-circuit networks: a fixed-length label. The goal was not to abandon the destination-based IP datagram-forwarding infrastructure for one based on fixed-length labels and virtual circuits, but to augment it by selectively labeling datagrams and allowing routers to forward datagrams based on fixed-length labels (rather than destination IP addresses) when possible. Importantly, these techniques work hand-in-hand with IP, using IP addressing and routing. The IETF unified these efforts in the MPLS protocol [RFC 3031, RFC 3032], effectively blending VC techniques into a routed datagram network.

Let's begin our study of MPLS by considering the format of a link-layer frame that is handled by an MPLS-capable router. Figure 6.28 shows that a link-layer frame transmitted between MPLS-capable devices has a small MPLS header added between the layer-2 (e.g., Ethernet) header and layer-3 (i.e., IP) header. RFC 3032 defines the format of the MPLS header for such links; headers are defined for ATM and frame-relayed networks as well in other RFCs. Among the fields in the MPLS

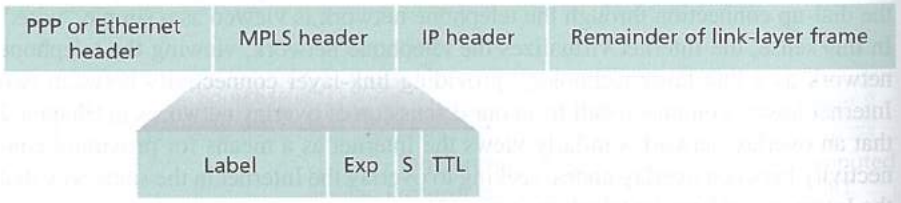
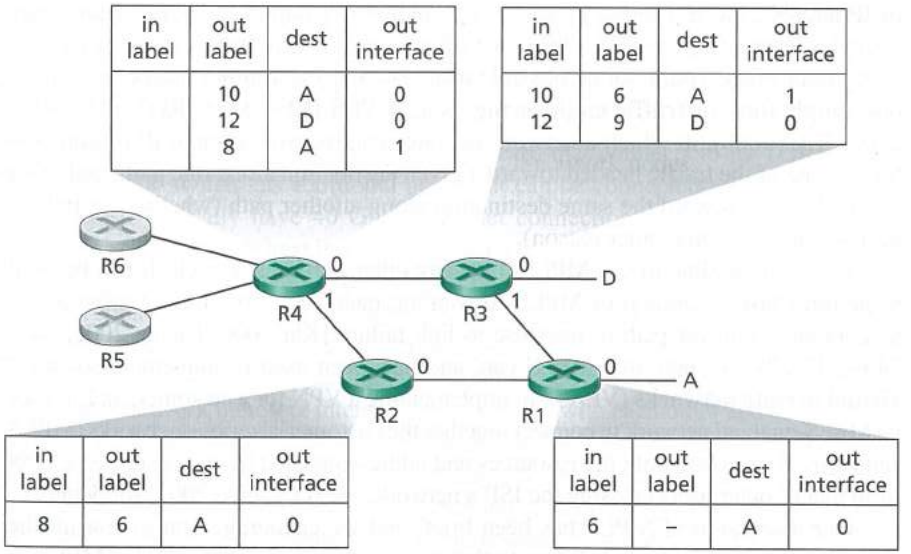


Figure 6.28 ♦ MPLS header: Located between link- and network-layer headers

header are the label, 3 bits reserved for experimental use, a single S bit, which is used to indicate the end of a series of "stacked" MPLS headers (an advanced topic that we'll not cover here), and a time-to-live field.

It's immediately evident from Figure 6.28 that an MPLS-enhanced frame can only be sent between routers that are both MPLS capable (since a non-MPLS-capable router would be quite confused when it found an MPLS header where it had expected to find the IP header!). An MPLS-capable router is often referred to as a **label-switched router**, since it forwards an MPLS frame by looking up the MPLS label in its forwarding table and then immediately passing the datagram to the appropriate output interface. Thus, the MPLS-capable router need *not* extract the destination IP address and perform a lookup of the longest prefix match in the forwarding table. But how does a router know if its neighbor is indeed MPLS capable, and how does a router know what label to associate with the given IP destination? To answer these questions, we'll need to take a look at the interaction among a group of MPLS-capable routers.

In the example in Figure 6.29, routers R1 through R4 are MPLS capable. R5 and R6 are standard IP routers. R1 has advertised to R2 and R3 that it (R1) can route to destination A, and that a received frame with MPLS label 6 will be forwarded to destination A. Router R3 has advertised to router R4 that it can route to destinations A and D, and that incoming frames with MPLS labels 10 and 12, respectively, will be switched toward those destinations. Router R2 has also advertised to router R4 that it (R2) can reach destination A, and that a received frame with MPLS label 8 will be switched toward A. Note that router R4 is now in the interesting position of having



in label	out label	dest	out interface
	10	A	0
	12	D	0
	8	A	1

in label	out label	dest	out interface
10	6	A	1
12	9	D	0

in label	out label	dest	out interface
8	6	A	0

in label	out label	dest	out interface
6	–	A	0

Figure 6.29 ♦ MPLS-enhanced forwarding

two MPLS paths to reach A: via interface 0 with outbound MPLS label 10, and via interface 1 with an MPLS label of 8. The broad picture painted in Figure 6.29 is that IP devices R5, R6, A, and D are connected together via an MPLS infrastructure (MPLS-capable routers R1, R2, R3, and R4) in much the same way that a switched LAN or an ATM network can connect together IP devices. And like a switched LAN or ATM network, the MPLS-capable routers R1 through R4 do so *without ever touching the IP header of a packet*.

In our discussion above, we've not specified the specific protocol used to distribute labels among the MPLS-capable routers, as the details of this signaling are well beyond the scope of this book. We note, however, that the IETF working group on MPLS has specified in [RFC 3468] that an extension of the RSVP protocol, known as RSVP-TE [RFC 3209], will be the focus of its efforts for MPLS signaling. We've also not discussed how MPLS actually computes the paths for packets among MPLS capable routers, nor how it gathers link-state information (e.g., amount of link bandwidth unreserved by MPLS) to use in these path computations. Existing link-state routing algorithms (e.g., OSPF) have been extended to flood this information to MPLS-capable routers. Interestingly, the actual path computation algorithms are not standardized, and are currently vendor-specific.

Thus far, the emphasis of our discussion of MPLS has been on the fact that MPLS performs switching based on labels, without needing to consider the IP address of a packet. The true advantages of MPLS and the reason for current interest in MPLS, however, lie not in the potential increases in switching speeds, but rather in the new traffic management capabilities that MPLS enables. As noted above, R4 has *two* MPLS paths to A. If forwarding were performed up at the IP layer on the basis of IP address, the IP routing protocols we studied in Chapter 5 would specify only a single, least-cost path to A. Thus, MPLS provides the ability to forward packets along routes that would not be possible using standard IP routing protocols. This is one simple form of **traffic engineering** using MPLS [RFC 3346; RFC 3272; RFC 2702; Xiao 2000], in which a network operator can override normal IP routing and force some of the traffic headed toward a given destination along one path, and other traffic destined toward the same destination along another path (whether for policy, performance, or some other reason).

It is also possible to use MPLS for many other purposes as well. It can be used to perform fast restoration of MPLS forwarding paths, e.g., to reroute traffic over a precomputed failover path in response to link failure [Kar 2000; Huang 2002; RFC 3469]. Finally, we note that MPLS can, and has, been used to implement so-called **virtual private networks** (VPNs). In implementing a VPN for a customer, an ISP uses its MPLS-enabled network to connect together the customer's various networks. MPLS can be used to isolate both the resources and addressing used by the customer's VPN from that of other users crossing the ISP's network; see [DeClercq 2002] for details.

Our discussion of MPLS has been brief, and we encourage you to consult the references we've mentioned. We note that with so many possible uses for MPLS, it appears that it is rapidly becoming the Swiss Army knife of Internet traffic engineering!

6.6 Data Center Networking

In recent years, Internet companies such as Google, Microsoft, Facebook, and Amazon (as well as their counterparts in Asia and Europe) have built massive data centers, each housing tens to hundreds of thousands of hosts, and concurrently supporting many distinct cloud applications (e.g., search, e-mail, social networking, and e-commerce). Each data center has its own **data center network** that interconnects its hosts with each other and interconnects the data center with the Internet. In this section, we provide a brief introduction to data center networking for cloud applications.

The cost of a large data center is huge, exceeding $12 million per month for a 100,000 host data center [Greenberg 2009a]. Of these costs, about 45 percent can be attributed to the hosts themselves (which need to be replaced every 3–4 years); 25 percent to infrastructure, including transformers, uninterruptable power supplies (UPS) systems, generators for long-term outages, and cooling systems; 15 percent for electric utility costs for the power draw; and 15 percent for networking, including network gear (switches, routers and load balancers), external links, and transit traffic costs. (In these percentages, costs for equipment are amortized so that a common cost metric is applied for one-time purchases and ongoing expenses such as power.) While networking is not the largest cost, networking innovation is the key to reducing overall cost and maximizing performance [Greenberg 2009a].

The worker bees in a data center are the hosts: They serve content (e.g., Web pages and videos), store e-mails and documents, and collectively perform massively distributed computations (e.g., distributed index computations for search engines). The hosts in data centers, called **blades** and resembling pizza boxes, are generally commodity hosts that include CPU, memory, and disk storage. The hosts are stacked in racks, with each rack typically having 20 to 40 blades. At the top of each rack there is a switch, aptly named the **Top of Rack (TOR) switch**, that interconnects the hosts in the rack with each other and with other switches in the data center. Specifically, each host in the rack has a network interface card that connects to its TOR switch, and each TOR switch has additional ports that can be connected to other switches. Today hosts typically have 40 Gbps Ethernet connections to their TOR switches [Greenberg 2015]. Each host is also assigned its own data-center-internal IP address.

The data center network supports two types of traffic: traffic flowing between external clients and internal hosts and traffic flowing between internal hosts. To handle flows between external clients and internal hosts, the data center network includes one or more **border routers**, connecting the data center network to the public Internet. The data center network therefore interconnects the racks with each other and connects the racks to the border routers. Figure 6.30 shows an example of a data center network. **Data center network design**, the art of designing the interconnection network and protocols that connect the racks with each other and with the border routers, has become an important branch of computer networking research in recent years [Al-Fares 2008; Greenberg 2009a; Greenberg 2009b; Mysore 2009; Guo 2009; Wang 2010].

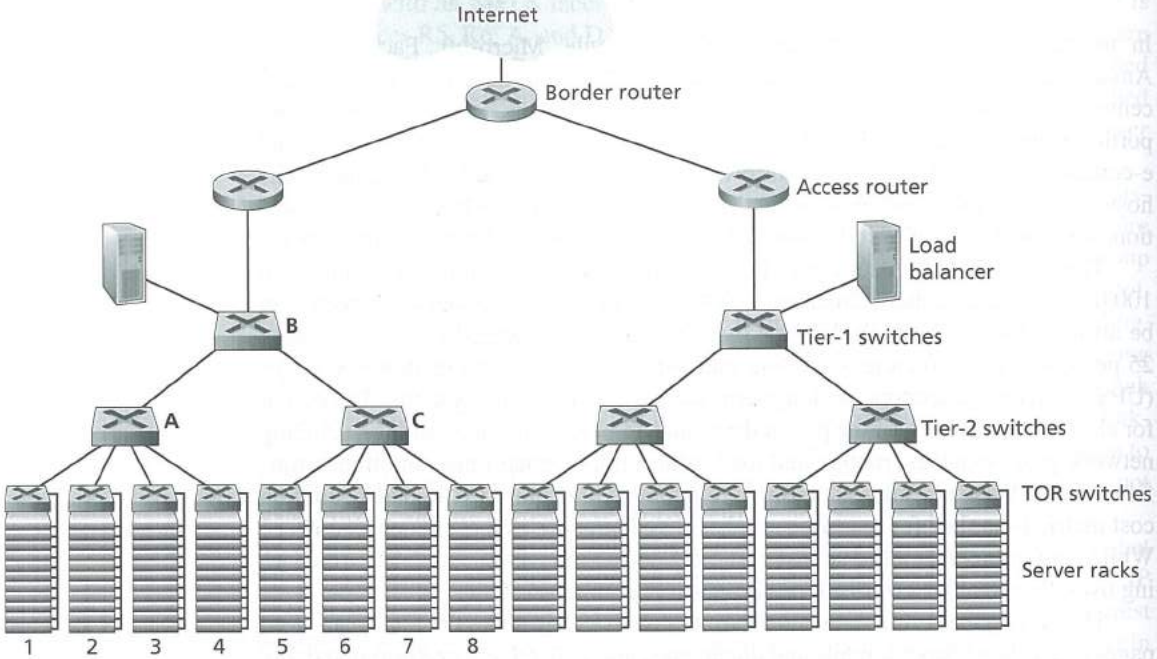


Figure 6.30 ♦ A data center network with a hierarchical topology

Load Balancing

A cloud data center, such as a Google or Microsoft data center, provides many applications concurrently, such as search, e-mail, and video applications. To support requests from external clients, each application is associated with a publicly visible IP address to which clients send their requests and from which they receive responses. Inside the data center, the external requests are first directed to a **load balancer** whose job it is to distribute requests to the hosts, balancing the load across the hosts as a function of their current load. A large data center will often have several load balancers, each one devoted to a set of specific cloud applications. Such a load balancer is sometimes referred to as a "layer-4 switch" since it makes decisions based on the destination port number (layer 4) as well as destination IP address in the packet. Upon receiving a request for a particular application, the load balancer forwards it to one of the hosts that handles the application. (A host may then invoke the services of other hosts to help process the request.) When the host finishes processing the request, it sends its response back to the load balancer, which in turn relays the response back to the external client. The load balancer not only balances

the work load across hosts, but also provides a NAT-like function, translating the public external IP address to the internal IP address of the appropriate host, and then translating back for packets traveling in the reverse direction back to the clients. This prevents clients from contacting hosts directly, which has the security benefit of hiding the internal network structure and preventing clients from directly interacting with the hosts.

Hierarchical Architecture

For a small data center housing only a few thousand hosts, a simple network consisting of a border router, a load balancer, and a few tens of racks all interconnected by a single Ethernet switch could possibly suffice. But to scale to tens to hundreds of thousands of hosts, a data center often employs a **hierarchy of routers and switches**, such as the topology shown in Figure 6.30. At the top of the hierarchy, the border router connects to access routers (only two are shown in Figure 6.30, but there can be many more). Below each access router there are three tiers of switches. Each access router connects to a top-tier switch, and each top-tier switch connects to multiple second-tier switches and a load balancer. Each second-tier switch in turn connects to multiple racks via the racks' TOR switches (third-tier switches). All links typically use Ethernet for their link-layer and physical-layer protocols, with a mix of copper and fiber cabling. With such a hierarchical design, it is possible to scale a data center to hundreds of thousands of hosts.

Because it is critical for a cloud application provider to continually provide applications with high availability, data centers also include redundant network equipment and redundant links in their designs (not shown in Figure 6.30). For example, each TOR switch can connect to two tier-2 switches, and each access router, tier-1 switch, and tier-2 switch can be duplicated and integrated into the design [Cisco 2012; Greenberg 2009b]. In the hierarchical design in Figure 6.30, observe that the hosts below each access router form a single subnet. In order to localize ARP broadcast traffic, each of these subnets is further partitioned into smaller VLAN subnets, each comprising a few hundred hosts [Greenberg 2009a].

Although the conventional hierarchical architecture just described solves the problem of scale, it suffers from *limited host-to-host capacity* [Greenberg 2009b]. To understand this limitation, consider again Figure 6.30, and suppose each host connects to its TOR switch with a 1 Gbps link, whereas the links between switches are 10 Gbps Ethernet links. Two hosts in the same rack can always communicate at a full 1 Gbps, limited only by the rate of the hosts' network interface cards. However, if there are many simultaneous flows in the data center network, the maximum rate between two hosts in *different* racks can be much less. To gain insight into this issue, consider a traffic pattern consisting of 40 simultaneous flows between 40 pairs of hosts in different racks. Specifically, suppose each of 10 hosts in rack 1 in Figure 6.30 sends a flow to a corresponding host in rack 5. Similarly, there are ten simultaneous flows between pairs of hosts in racks 2 and 6, ten simultaneous flows

between racks 3 and 7, and ten simultaneous flows between racks 4 and 8. If each flow evenly shares a link's capacity with other flows traversing that link, then the 40 flows crossing the 10 Gbps A-to-B link (as well as the 10 Gbps B-to-C link) will each only receive 10 Gbps / 40 = 250 Mbps, which is significantly less than the 1 Gbps network interface card rate. The problem becomes even more acute for flows between hosts that need to travel higher up the hierarchy. One possible solution to this limitation is to deploy higher-rate switches and routers. But this would significantly increase the cost of the data center, because switches and routers with high port speeds are very expensive.

Supporting high-bandwidth host-to-host communication is important because a key requirement in data centers is flexibility in placement of computation and services [Greenberg 2009b; Farrington 2010]. For example, a large-scale Internet search engine may run on thousands of hosts spread across multiple racks with significant bandwidth requirements between all pairs of hosts. Similarly, a cloud computing service such as EC2 may wish to place the multiple virtual machines comprising a customer's service on the physical hosts with the most capacity irrespective of their location in the data center. If these physical hosts are spread across multiple racks, network bottlenecks as described above may result in poor performance.

Trends in Data Center Networking

In order to reduce the cost of data centers, and at the same time improve their delay and throughput performance, Internet cloud giants such as Google, Facebook, Amazon, and Microsoft are continually deploying new data center network designs. Although these designs are proprietary, many important trends can nevertheless be identified.

One such trend is to deploy new interconnection architectures and network protocols that overcome the drawbacks of the traditional hierarchical designs. One such approach is to replace the hierarchy of switches and routers with a **fully connected topology** [Facebook 2014; Al-Fares 2008; Greenberg 2009b; Guo 2009], such as the topology shown in Figure 6.31. In this design, each tier-1 switch connects to all of the tier-2 switches so that (1) host-to-host traffic never has to rise above the switch tiers, and (2) with n tier-1 switches, between any two tier-2 switches there are n disjoint paths. Such a design can significantly improve the host-to-host capacity. To see this, consider again our example of 40 flows. The topology in Figure 6.31 can handle such a flow pattern since there are four distinct paths between the first tier-2 switch and the second tier-2 switch, together providing an aggregate capacity of 40 Gbps between the first two tier-2 switches. Such a design not only alleviates the host-to-host capacity limitation, but also creates a more flexible computation and service environment in which communication between any two racks not connected to the same switch is logically equivalent, irrespective of their locations in the data center.

Another major trend is to employ shipping container–based modular data centers (MDCs) [YouTube 2009; Waldrop 2007]. In an MDC, a factory builds, within a

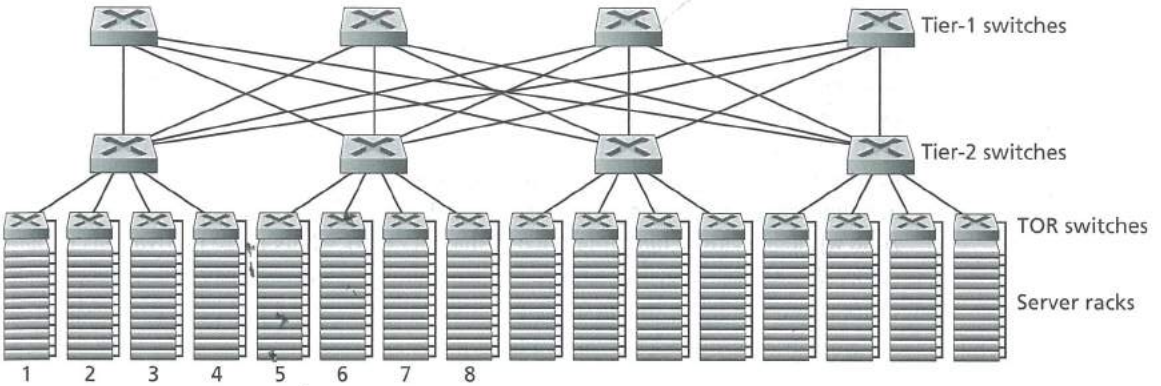


Figure 6.31 ♦ Highly interconnected data network topology

standard 12-meter shipping container, a "mini data center" and ships the container to the data center location. Each container has up to a few thousand hosts, stacked in tens of racks, which are packed closely together. At the data center location, multiple containers are interconnected with each other and also with the Internet. Once a prefabricated container is deployed at a data center, it is often difficult to service. Thus, each container is designed for graceful performance degradation: as components (servers and switches) fail over time, the container continues to operate but with degraded performance. When many components have failed and performance has dropped below a threshold, the entire container is removed and replaced with a fresh one.

Building a data center out of containers creates new networking challenges. With an MDC, there are two types of networks: the container-internal networks within each of the containers and the core network connecting each container [Guo 2009; Farrington 2010]. Within each container, at the scale of up to a few thousand hosts, it is possible to build a fully connected network (as described above) using inexpensive commodity Gigabit Ethernet switches. However, the design of the core network, interconnecting hundreds to thousands of containers while providing high host-to-host bandwidth across containers for typical workloads, remains a challenging problem. A hybrid electrical/optical switch architecture for interconnecting the containers is proposed in [Farrington 2010].

When using highly interconnected topologies, one of the major issues is designing routing algorithms among the switches. One possibility [Greenberg 2009b] is to use a form of random routing. Another possibility [Guo 2009] is to deploy multiple network interface cards in each host, connect each host to multiple low-cost commodity switches, and allow the hosts themselves to intelligently route traffic among the switches. Variations and extensions of these approaches are currently being deployed in contemporary data centers.

Another important trend is that large cloud providers are increasingly building or customizing just about everything that is in their data centers, including network adapters, switches routers, TORs, software, and networking protocols [Greenberg 2015, Singh 2015]. Another trend, pioneered by Amazon, is to improve reliability with "availability zones," which essentially replicate distinct data centers in different nearby buildings. By having the buildings nearby (a few kilometers apart), transactional data can be synchronized across the data centers in the same availability zone while providing fault tolerance [Amazon 2014]. Many more innovations in data center design are likely to continue to come; interested readers are encouraged to see the recent papers and videos on data center network design.

6.7 Retrospective: A Day in the Life of a Web Page Request

Now that we've covered the link layer in this chapter, and the network, transport and application layers in earlier chapters, our journey down the protocol stack is complete! In the very beginning of this book (Section 1.1), we wrote "much of this book is concerned with computer network protocols," and in the first five chapters, we've certainly seen that this is indeed the case! Before heading into the topical chapters in second part of this book, we'd like to wrap up our journey down the protocol stack by taking an integrated, holistic view of the protocols we've learned about so far. One way then to take this "big picture" view is to identify the many (many!) protocols that are involved in satisfying even the simplest request: downloading a Web page. Figure 6.32 illustrates our setting: a student, Bob, connects a laptop to his school's Ethernet switch and downloads a Web page (say the home page of www.google.com). As we now know, there's a *lot* going on "under the hood" to satisfy this seemingly simple request. A Wireshark lab at the end of this chapter examines trace files containing a number of the packets involved in similar scenarios in more detail.

6.7.1 Getting Started: DHCP, UDP, IP, and Ethernet

Let's suppose that Bob boots up his laptop and then connects it to an Ethernet cable connected to the school's Ethernet switch, which in turn is connected to the school's router, as shown in Figure 6.32. The school's router is connected to an ISP, in this example, comcast.net. In this example, comcast.net is providing the DNS service for the school; thus, the DNS server resides in the Comcast network rather than the school network. We'll assume that the DHCP server is running within the router, as is often the case.

When Bob first connects his laptop to the network, he can't do anything (e.g., download a Web page) without an IP address. Thus, the first network-related

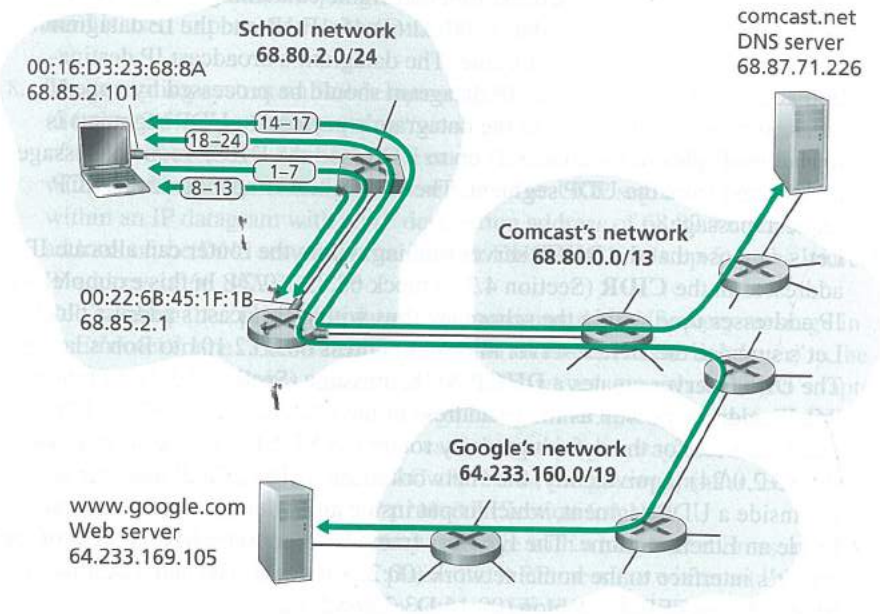


Figure 6.32 ♦ A day in the life of a Web page request: Network setting and actions

action taken by Bob's laptop is to run the DHCP protocol to obtain an IP address, as well as other information, from the local DHCP server:

1. The operating system on Bob's laptop creates a **DHCP request message** (Section 4.3.3) and puts this message within a **UDP segment** (Section 3.3) with destination port 67 (DHCP server) and source port 68 (DHCP client). The UDP segment is then placed within an **IP datagram** (Section 4.3.1) with a broadcast IP destination address (255.255.255.255) and a source IP address of 0.0.0.0, since Bob's laptop doesn't yet have an IP address.
2. The IP datagram containing the DHCP request message is then placed within an **Ethernet frame** (Section 6.4.2). The Ethernet frame has a destination MAC addresses of FF:FF:FF:FF:FF:FF so that the frame will be broadcast to all devices connected to the switch (hopefully including a DHCP server); the frame's source MAC address is that of Bob's laptop, 00:16:D3:23:68:8A.
3. The broadcast Ethernet frame containing the DHCP request is the first frame sent by Bob's laptop to the Ethernet switch. The switch broadcasts the incoming frame on all outgoing ports, including the port connected to the router.

4. The router receives the broadcast Ethernet frame containing the DHCP request on its interface with MAC address 00:22:6B:45:1F:1B and the IP datagram is extracted from the Ethernet frame. The datagram's broadcast IP destination address indicates that this IP datagram should be processed by upper layer protocols at this node, so the datagram's payload (a UDP segment) is thus **demultiplexed** (Section 3.2) up to UDP, and the DHCP request message is extracted from the UDP segment. The DHCP server now has the DHCP request message.
5. Let's suppose that the DHCP server running within the router can allocate IP addresses in the **CIDR** (Section 4.3.3) block 68.85.2.0/24. In this example, all IP addresses used within the school are thus within Comcast's address block. Let's suppose the DHCP server allocates address 68.85.2.101 to Bob's laptop. The DHCP server creates a **DHCP ACK message** (Section 4.3.3) containing this IP address, as well as the IP address of the DNS server (68.87.71.226), the IP address for the default gateway router (68.85.2.1), and the subnet block (68.85.2.0/24) (equivalently, the "network mask"). The DHCP message is put inside a UDP segment, which is put inside an IP datagram, which is put inside an Ethernet frame. The Ethernet frame has a source MAC address of the router's interface to the home network (00:22:6B:45:1F:1B) and a destination MAC address of Bob's laptop (00:16:D3:23:68:8A).
6. The Ethernet frame containing the DHCP ACK is sent (unicast) by the router to the switch. Because the switch is **self-learning** (Section 6.4.3) and previously received an Ethernet frame (containing the DHCP request) from Bob's laptop, the switch knows to forward a frame addressed to 00:16:D3:23:68:8A only to the output port leading to Bob's laptop.
7. Bob's laptop receives the Ethernet frame containing the DHCP ACK, extracts the IP datagram from the Ethernet frame, extracts the UDP segment from the IP datagram, and extracts the DHCP ACK message from the UDP segment. Bob's DHCP client then records its IP address and the IP address of its DNS server. It also installs the address of the default gateway into its **IP forwarding table** (Section 4.1). Bob's laptop will send all datagrams with destination address outside of its subnet 68.85.2.0/24 to the default gateway. At this point, Bob's laptop has initialized its networking components and is ready to begin processing the Web page fetch. (Note that only the last two DHCP steps of the four presented in Chapter 4 are actually necessary.)

6.7.2 Still Getting Started: DNS and ARP

When Bob types the URL for www.google.com into his Web browser, he begins the long chain of events that will eventually result in Google's home page being displayed by his Web browser. Bob's Web browser begins the process by creating a **TCP socket** (Section 2.7) that will be used to send the **HTTP request** (Section 2.2) to www.google.com. In order to create the socket, Bob's laptop will need to

know the IP address of www.google.com. We learned in Section 2.5, that the **DNS protocol** is used to provide this name-to-IP-address translation service.

8. The operating system on Bob's laptop thus creates a **DNS query message** (Section 2.5.3), putting the string "www.google.com" in the question section of the DNS message. This DNS message is then placed within a UDP segment with a destination port of 53 (DNS server). The UDP segment is then placed within an IP datagram with an IP destination address of 68.87.71.226 (the address of the DNS server returned in the DHCP ACK in step 5) and a source IP address of 68.85.2.101.
9. Bob's laptop then places the datagram containing the DNS query message in an Ethernet frame. This frame will be sent (addressed, at the link layer) to the gateway router in Bob's school's network. However, even though Bob's laptop knows the IP address of the school's gateway router (68.85.2.1) via the DHCP ACK message in step 5 above, it doesn't know the gateway router's MAC address. In order to obtain the MAC address of the gateway router, Bob's laptop will need to use the **ARP protocol** (Section 6.4.1).
10. Bob's laptop creates an **ARP query** message with a target IP address of 68.85.2.1 (the default gateway), places the ARP message within an Ethernet frame with a broadcast destination address (FF:FF:FF:FF:FF:FF) and sends the Ethernet frame to the switch, which delivers the frame to all connected devices, including the gateway router.
11. The gateway router receives the frame containing the ARP query message on the interface to the school network, and finds that the target IP address of 68.85.2.1 in the ARP message matches the IP address of its interface. The gateway router thus prepares an **ARP reply**, indicating that its MAC address of 00:22:6B:45:1F:1B corresponds to IP address 68.85.2.1. It places the ARP reply message in an Ethernet frame, with a destination address of 00:16:D3:23:68:8A (Bob's laptop) and sends the frame to the switch, which delivers the frame to Bob's laptop.
12. Bob's laptop receives the frame containing the ARP reply message and extracts the MAC address of the gateway router (00:22:6B:45:1F:1B) from the ARP reply message.
13. Bob's laptop can now (*finally!*) address the Ethernet frame containing the DNS query to the gateway router's MAC address. Note that the IP datagram in this frame has an IP destination address of 68.87.71.226 (the DNS server), while the frame has a destination address of 00:22:6B:45:1F:1B (the gateway router). Bob's laptop sends this frame to the switch, which delivers the frame to the gateway router.

6.7.3 Still Getting Started: Intra-Domain Routing to the DNS Server

14. The gateway router receives the frame and extracts the IP datagram containing the DNS query. The router looks up the destination address of this datagram

(68.87.71.226) and determines from its forwarding table that the datagram should be sent to the leftmost router in the Comcast network in Figure 6.32. The IP datagram is placed inside a link-layer frame appropriate for the link connecting the school's router to the leftmost Comcast router and the frame is sent over this link.

15. The leftmost router in the Comcast network receives the frame, extracts the IP datagram, examines the datagram's destination address (68.87.71.226) and determines the outgoing interface on which to forward the datagram toward the DNS server from its forwarding table, which has been filled in by Comcast's intra-domain protocol (such as **RIP**, **OSPF** or **IS-IS**, Section 5.3) as well as the **Internet's inter-domain protocol**, **BGP** (Section 5.4).
16. Eventually the IP datagram containing the DNS query arrives at the DNS server. The DNS server extracts the DNS query message, looks up the name www.google.com in its DNS database (Section 2.5), and finds the **DNS resource record** that contains the IP address (64.233.169.105) for www.google.com. (assuming that it is currently cached in the DNS server). Recall that this cached data originated in the **authoritative DNS server** (Section 2.5.2) for googlecom. The DNS server forms a **DNS reply message** containing this hostname-to-IP-address mapping, and places the DNS reply message in a UDP segment, and the segment within an IP datagram addressed to Bob's laptop (68.85.2.101). This datagram will be forwarded back through the Comcast network to the school's router and from there, via the Ethernet switch to Bob's laptop.
17. Bob's laptop extracts the IP address of the server www.google.com from the DNS message. *Finally*, after a *lot* of work, Bob's laptop is now ready to contact the www.google.com server!

6.7.4 Web Client-Server Interaction: TCP and HTTP

18. Now that Bob's laptop has the IP address of www.google.com, it can create the **TCP socket** (Section 2.7) that will be used to send the **HTTP GET** message (Section 2.2.3) to www.google.com. When Bob creates the TCP socket, the TCP in Bob's laptop must first perform a **three-way handshake** (Section 3.5.6) with the TCP in www.google.com. Bob's laptop thus first creates a **TCP SYN** segment with destination port 80 (for HTTP), places the TCP segment inside an IP datagram with a destination IP address of 64.233.169.105 (www.google.com), places the datagram inside a frame with a destination MAC address of 00:22:6B:45:1F:1B (the gateway router) and sends the frame to the switch.
19. The routers in the school network, Comcast's network, and Google's network forward the datagram containing the TCP SYN toward www.google.com, using the forwarding table in each router, as in steps 14–16 above. Recall that the router forwarding table entries governing forwarding of packets over the inter-domain link between the Comcast and Google networks are determined by the **BGP** protocol (Chapter 5).

20. Eventually, the datagram containing the TCP SYN arrives at www.google .com. The TCP SYN message is extracted from the datagram and demultiplexed to the welcome socket associated with port 80. A connection socket (Section 2.7) is created for the TCP connection between the Google HTTP server and Bob's laptop. A TCP SYNACK (Section 3.5.6) segment is generated, placed inside a datagram addressed to Bob's laptop, and finally placed inside a link-layer frame appropriate for the link connecting www.google.com to its first-hop router.
21. The datagram containing the TCP SYNACK segment is forwarded through the Google, Comcast, and school networks, eventually arriving at the Ethernet card in Bob's laptop. The datagram is demultiplexed within the operating system to the TCP socket created in step 18, which enters the connected state.
22. With the socket on Bob's laptop now (*finally!*) ready to send bytes to www .google.com, Bob's browser creates the HTTP GET message (Section 2.2.3) containing the URL to be fetched. The HTTP GET message is then written into the socket, with the GET message becoming the payload of a TCP segment. The TCP segment is placed in a datagram and sent and delivered to www .google.com as in steps 18–20 above.
23. The HTTP server at www.google.com reads the HTTP GET message from the TCP socket, creates an **HTTP response** message (Section 2.2), places the requested Web page content in the body of the HTTP response message, and sends the message into the TCP socket.
24. The datagram containing the HTTP reply message is forwarded through the Google, Comcast, and school networks, and arrives at Bob's laptop. Bob's Web browser program reads the HTTP response from the socket, extracts the html for the Web page from the body of the HTTP response, and finally (*finally!*) displays the Web page!

Our scenario above has covered a lot of networking ground! If you've understood most or all of the above example, then you've also covered a lot of ground since you first read Section 1.1, where we wrote "much of this book is concerned with computer network protocols" and you may have wondered what a protocol actually was! As detailed as the above example might seem, we've omitted a number of possible additional protocols (e.g., NAT running in the school's gateway router, wireless access to the school's network, security protocols for accessing the school network or encrypting segments or datagrams, network management protocols), and considerations (Web caching, the DNS hierarchy) that one would encounter in the public Internet. We'll cover a number of these topics and more in the second part of this book.

Lastly, we note that our example above was an integrated and holistic, but also very "nuts and bolts," view of many of the protocols that we've studied in the first part of this book. The example focused more on the "how" than the "why." For a broader, more reflective view on the design of network protocols in general, see [Clark 1988, RFC 5218].

6.8 Summary

In this chapter, we've examined the link layer—its services, the principles underlying its operation, and a number of important specific protocols that use these principles in implementing link-layer services.

We saw that the basic service of the link layer is to move a network-layer datagram from one node (host, switch, router, WiFi access point) to an adjacent node. We saw that all link-layer protocols operate by encapsulating a network-layer datagram within a link-layer frame before transmitting the frame over the link to the adjacent node. Beyond this common framing function, however, we learned that different link-layer protocols provide very different link access, delivery, and transmission services. These differences are due in part to the wide variety of link types over which link-layer protocols must operate. A simple point-to-point link has a single sender and receiver communicating over a single "wire." A multiple access link is shared among many senders and receivers; consequently, the link-layer protocol for a multiple access channel has a protocol (its multiple access protocol) for coordinating link access. In the case of MPLS, the "link" connecting two adjacent nodes (for example, two IP routers that are adjacent in an IP sense—that they are next-hop IP routers toward some destination) may actually be a *network* in and of itself. In one sense, the idea of a network being considered as a link should not seem odd. A telephone link connecting a home modem/computer to a remote modem/router, for example, is actually a path through a sophisticated and complex telephone *network*.

Among the principles underlying link-layer communication, we examined error-detection and -correction techniques, multiple access protocols, link-layer addressing, virtualization (VLANs), and the construction of extended switched LANs and data center networks. Much of the focus today at the link layer is on these switched networks. In the case of error detection/correction, we examined how it is possible to add additional bits to a frame's header in order to detect, and in some cases correct, bit-flip errors that might occur when the frame is transmitted over the link. We covered simple parity and checksumming schemes, as well as the more robust cyclic redundancy check. We then moved on to the topic of multiple access protocols. We identified and studied three broad approaches for coordinating access to a broadcast channel: channel partitioning approaches (TDM, FDM), random access approaches (the ALOHA protocols and CSMA protocols), and taking-turns approaches (polling and token passing). We studied the cable access network and found that it uses many of these multiple access methods. We saw that a consequence of having multiple nodes share a single broadcast channel was the need to provide node addresses at the link layer. We learned that link-layer addresses were quite different from network-layer addresses and that, in the case of the Internet, a special protocol (ARP—the Address Resolution Protocol) is used to translate between these two forms of addressing and studied the hugely successful Ethernet protocol in detail. We then examined how nodes sharing a broadcast channel form a LAN and how multiple LANs can be connected together to form larger LANs—all *without* the intervention

of network-layer routing to interconnect these local nodes. We also learned how multiple virtual LANs can be created on a single physical LAN infrastructure.

We ended our study of the link layer by focusing on how MPLS networks provide link-layer services when they interconnect IP routers and an overview of the network designs for today's massive data centers. We wrapped up this chapter (and indeed the first five chapters) by identifying the many protocols that are needed to fetch a simple Web page. Having covered the link layer, *our journey down the protocol stack is now over*! Certainly, the physical layer lies below the link layer, but the details of the physical layer are probably best left for another course (for example, in communication theory, rather than computer networking). We have, however, touched upon several aspects of the physical layer in this chapter and in Chapter 1 (our discussion of physical media in Section 1.2). We'll consider the physical layer again when we study wireless link characteristics in the next chapter.

Although our journey down the protocol stack is over, our study of computer networking is not yet at an end. In the following three chapters we cover wireless networking, network security, and multimedia networking. These four topics do not fit conveniently into any one layer; indeed, each topic crosscuts many layers. Understanding these topics (billed as advanced topics in some networking texts) thus requires a firm foundation in all layers of the protocol stack—a foundation that our study of the link layer has now completed!

Homework Problems and Questions

Chapter 6 Review Questions

SECTIONS 6.1–6.2

R1. Consider the transportation analogy in Section 6.1.1. If the passenger is analagous to a datagram, what is analogous to the link layer frame?

R2. If all the links in the Internet were to provide reliable delivery service, would the TCP reliable delivery service be redundant? Why or why not?

R3. What are some of the possible services that a link-layer protocol can offer to the network layer? Which of these link-layer services have corresponding services in IP? In TCP?

SECTION 6.3

R4. Suppose two nodes start to transmit at the same time a packet of length L over a broadcast channel of rate R. Denote the propagation delay between the two nodes as d_{prop}. Will there be a collision if $d_{\text{prop}} < L/R$? Why or why not?

R5. In Section 6.3, we listed four desirable characteristics of a broadcast channel. Which of these characteristics does slotted ALOHA have? Which of these characteristics does token passing have?

R6. In CSMA/CD, after the fifth collision, what is the probability that a node chooses $K = 4$? The result $K = 4$ corresponds to a delay of how many seconds on a 10 Mbps Ethernet?

R7. Describe polling and token-passing protocols using the analogy of cocktail party interactions.

R8. Why would the token-ring protocol be inefficient if a LAN had a very large perimeter?

SECTION 6.4

R9. How big is the MAC address space? The IPv4 address space? The IPv6 address space?

R10. Suppose nodes A, B, and C each attach to the same broadcast LAN (through their adapters). If A sends thousands of IP datagrams to B with each encapsulating frame addressed to the MAC address of B, will C's adapter process these frames? If so, will C's adapter pass the IP datagrams in these frames to the network layer C? How would your answers change if A sends frames with the MAC broadcast address?

R11. Why is an ARP query sent within a broadcast frame? Why is an ARP response sent within a frame with a specific destination MAC address?

R12. For the network in Figure 6.19, the router has two ARP modules, each with its own ARP table. Is it possible that the same MAC address appears in both tables?

R13. Compare the frame structures for 10BASE-T, 100BASE-T, and Gigabit Ethernet. How do they differ?

R14. Consider Figure 6.15. How many subnetworks are there, in the addressing sense of Section 4.3?

R15. What is the maximum number of VLANs that can be configured on a switch supporting the 802.1Q protocol? Why?

R16. Suppose that N switches supporting K VLAN groups are to be connected via a trunking protocol. How many ports are needed to connect the switches? Justify your answer.

Problems

P1. Suppose the information content of a packet is the bit pattern 1110 0110 1001 1101 and an even parity scheme is being used. What would the value of the field containing the parity bits be for the case of a two-dimensional parity scheme? Your answer should be such that a minimum-length checksum field is used.

P2. Show (give an example other than the one in Figure 6.5) that two-dimensional parity checks can correct and detect a single bit error. Show (give an example of) a double-bit error that can be detected but not corrected.

P3. Suppose the information portion of a packet (D in Figure 6.3) contains 10 bytes consisting of the 8-bit unsigned binary ASCII representation of string "Networking." Compute the Internet checksum for this data.

P4. Consider the previous problem, but instead suppose these 10 bytes contain

a. the binary representation of the numbers 1 through 10.

b. the ASCII representation of the letters B through K (uppercase).

c. the ASCII representation of the letters b through k (lowercase).

d. Compute the Internet checksum for this data.

P5. Consider the 5-bit generator, G = 10011, and suppose that D has the value 1010101010. What is the value of R?

P6. Consider the previous problem, but suppose that D has the value

a. 1001010101.

b. 0101101010.

c. 1010100000.

P7. In this problem, we explore some of the properties of the CRC. For the generator G (= 1001) given in Section 6.2.3, answer the following questions.

a. Why can it detect any single bit error in data D?

b. Can the above G detect any odd number of bit errors? Why?

P8. In Section 6.3, we provided an outline of the derivation of the efficiency of slotted ALOHA. In this problem we'll complete the derivation.

a. Recall that when there are N active nodes, the efficiency of slotted ALOHA is $Np(1 - p)^{N-1}$. Find the value of p that maximizes this expression.

b. Using the value of p found in (a), find the efficiency of slotted ALOHA by letting N approach infinity. *Hint*: $(1 - 1/N)^N$ approaches $1/e$ as N approaches infinity.

P9. Show that the maximum efficiency of pure ALOHA is $1/(2e)$. *Note*: This problem is easy if you have completed the problem above!

P10. Consider two nodes, A and B, that use the slotted ALOHA protocol to contend for a channel. Suppose node A has more data to transmit than node B, and node A's retransmission probability p_A is greater than node B's retransmission probability, p_B.

a. Provide a formula for node A's average throughput. What is the total efficiency of the protocol with these two nodes?

b. If $p_A = 2p_B$, is node A's average throughput twice as large as that of node B? Why or why not? If not, how can you choose p_A and p_B to make that happen?

c. In general, suppose there are N nodes, among which node A has retransmission probability $2p$ and all other nodes have retransmission probability p. Provide expressions to compute the average throughputs of node A and of any other node.

P11. Suppose four active nodes—nodes A, B, C and D—are competing for access to a channel using slotted ALOHA. Assume each node has an infinite number of packets to send. Each node attempts to transmit in each slot with probability p. The first slot is numbered slot 1, the second slot is numbered slot 2, and so on.

a. What is the probability that node A succeeds for the first time in slot 5?

b. What is the probability that some node (either A, B, C or D) succeeds in slot 4?

c. What is the probability that the first success occurs in slot 3?

d. What is the efficiency of this four-node system?

P12. Graph the efficiency of slotted ALOHA and pure ALOHA as a function of p for the following values of N:

a. $N = 15$.

b. $N = 25$.

c. $N = 35$.

P13. Consider a broadcast channel with N nodes and a transmission rate of R bps. Suppose the broadcast channel uses polling (with an additional polling node) for multiple access. Suppose the amount of time from when a node completes transmission until the subsequent node is permitted to transmit (that is, the polling delay) is d_{poll}. Suppose that within a polling round, a given node is allowed to transmit at most Q bits. What is the maximum throughput of the broadcast channel?

P14. Consider three LANs interconnected by two routers, as shown in Figure 6.33.

a. Assign IP addresses to all of the interfaces. For Subnet 1 use addresses of the form 192.168.1.xxx; for Subnet 2 uses addresses of the form 192.168.2.xxx; and for Subnet 3 use addresses of the form 192.168.3.xxx.

b. Assign MAC addresses to all of the adapters.

c. Consider sending an IP datagram from Host E to Host B. Suppose all of the ARP tables are up to date. Enumerate all the steps, as done for the single-router example in Section 6.4.1.

d. Repeat (c), now assuming that the ARP table in the sending host is empty (and the other tables are up to date).

P15. Consider Figure 6.33. Now we replace the router between subnets 1 and 2 with a switch S1, and label the router between subnets 2 and 3 as R1.

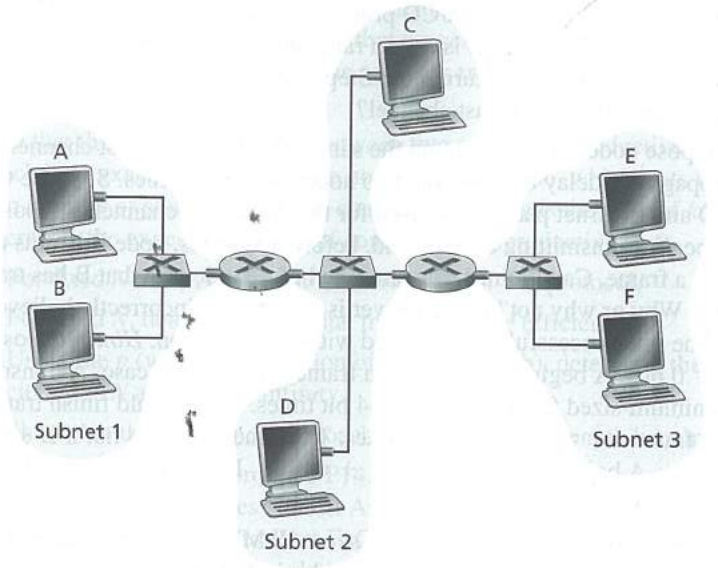


Figure 6.33 ♦ Three subnets, interconnected by routers

a. Consider sending an IP datagram from Host E to Host F. Will Host E ask router R1 to help forward the datagram? Why? In the Ethernet frame containing the IP datagram, what are the source and destination IP and MAC addresses?

b. Suppose E would like to send an IP datagram to B, and assume that E's ARP cache does not contain B's MAC address. Will E perform an ARP query to find B's MAC address? Why? In the Ethernet frame (containing the IP datagram destined to B) that is delivered to router R1, what are the source and destination IP and MAC addresses?

c. Suppose Host A would like to send an IP datagram to Host B, and neither A's ARP cache contains B's MAC address nor does B's ARP cache contain A's MAC address. Further suppose that the switch S1's forwarding table contains entries for Host B and router R1 only. Thus, A will broadcast an ARP request message. What actions will switch S1 perform once it receives the ARP request message? Will router R1 also receive this ARP request message? If so, will R1 forward the message to Subnet 3? Once Host B receives this ARP request message, it will send back to Host A an ARP response message. But will it send an ARP query message to ask for A's MAC address? Why? What will switch S1 do once it receives an ARP response message from Host B?

P16. Consider the previous problem, but suppose now that the router between subnets 2 and 3 is replaced by a switch. Answer questions (a)–(c) in the previous problem in this new context.

P17. Recall that with the CSMA/CD protocol, the adapter waits $K \cdot 512$ bit times after a collision, where K is drawn randomly. For $K = 100$, how long does the adapter wait until returning to Step 2 for a 10 Mbps broadcast channel? For a 100 Mbps broadcast channel?

P18. Suppose nodes A and B are on the same 10 Mbps broadcast channel, and the propagation delay between the two nodes is 325 bit times. Suppose CSMA/CD and Ethernet packets are used for this broadcast channel. Suppose node A begins transmitting a frame and, before it finishes, node B begins transmitting a frame. Can A finish transmitting before it detects that B has transmitted? Why or why not? If the answer is yes, then A incorrectly believes that its frame was successfully transmitted without a collision. *Hint*: Suppose at time $t = 0$ bits, A begins transmitting a frame. In the worst case, A transmits a minimum-sized frame of $512 + 64$ bit times. So A would finish transmitting the frame at $t = 512 + 64$ bit times. Thus, the answer is no, if B's signal reaches A before bit time $t = 512 + 64$ bits. In the worst case, when does B's signal reach A?

P19. Suppose nodes A and B are on the same 10 Mbps broadcast channel, and the propagation delay between the two nodes is 245 bit times. Suppose A and B send Ethernet frames at the same time, the frames collide, and then A and B choose different values of K in the CSMA/CD algorithm. Assuming no other nodes are active, can the retransmissions from A and B collide? For our purposes, it suffices to work out the following example. Suppose A and B begin transmission at $t = 0$ bit times. They both detect collisions at $t = 245$ t bit times. Suppose $K_A = 0$ and $K_B = 1$. At what time does B schedule its retransmission? At what time does A begin transmission? (*Note*: The nodes must wait for an idle channel after returning to Step 2—see protocol.) At what time does A's signal reach B? Does B refrain from transmitting at its scheduled time?

P20. In this problem, you will derive the efficiency of a CSMA/CD-like multiple access protocol. In this protocol, time is slotted and all adapters are synchronized to the slots. Unlike slotted ALOHA, however, the length of a slot (in seconds) is much less than a frame time (the time to transmit a frame). Let S be the length of a slot. Suppose all frames are of constant length $L = kRS$, where R is the transmission rate of the channel and k is a large integer. Suppose there are N nodes, each with an infinite number of frames to send. We also assume that $d_{\text{prop}} < S$, so that all nodes can detect a collision before the end of a slot time. The protocol is as follows:

- If, for a given slot, no node has possession of the channel, all nodes contend for the channel; in particular, each node transmits in the slot with probability p. If exactly one node transmits in the slot, that node takes possession of the channel for the subsequent $k - 1$ slots and transmits its entire frame.

- If some node has possession of the channel, all other nodes refrain from transmitting until the node that possesses the channel has finished transmitting its frame. Once this node has transmitted its frame, all nodes contend for the channel.

Note that the channel alternates between two states: the productive state, which lasts exactly k slots, and the nonproductive state, which lasts for a random number of slots. Clearly, the channel efficiency is the ratio of $k/(k + x)$, where x is the expected number of consecutive unproductive slots.

a. For fixed N and p, determine the efficiency of this protocol.

b. For fixed N, determine the p that maximizes the efficiency.

c. Using the p (which is a function of N) found in (b), determine the efficiency as N approaches infinity.

d. Show that this efficiency approaches 1 as the frame length becomes large.

P21. Consider Figure 6.33 in problem P14. Provide MAC addresses and IP addresses for the interfaces at Host A, both routers, and Host F. Suppose Host A sends a datagram to Host F. Give the source and destination MAC addresses in the frame encapsulating this IP datagram as the frame is transmitted *(i)* from A to the left router, *(ii)* from the left router to the right router, *(iii)* from the right router to F. Also give the source and destination IP addresses in the IP datagram encapsulated within the frame at each of these points in time.

P22. Suppose now that the leftmost router in Figure 6.33 is replaced by a switch. Hosts A, B, C, and D and the right router are all star-connected into this switch. Give the source and destination MAC addresses in the frame encapsulating this IP datagram as the frame is transmitted *(i)* from A to the switch, *(ii)* from the switch to the right router, *(iii)* from the right router to F. Also give the source and destination IP addresses in the IP datagram encapsulated within the frame at each of these points in time.

P23. Consider Figure 6.15. Suppose that all links are 100 Mbps. What is the maximum total aggregate throughput that can be achieved among the 9 hosts and 2 servers in this network? You can assume that any host or server can send to any other host or server. Why?

P24. Suppose the three departmental switches in Figure 6.15 are replaced by hubs. All links are 100 Mbps. Now answer the questions posed in problem P23.

P25. Suppose that *all* the switches in Figure 6.15 are replaced by hubs. All links are 100 Mbps. Now answer the questions posed in problem P23.

P26. Let's consider the operation of a learning switch in the context of a network in which 6 nodes labeled A through F are star connected into an Ethernet switch. Suppose that *(i)* B sends a frame to E, *(ii)* E replies with a frame to B, *(iii)* A sends a frame to B, *(iv)* B replies with a frame to A. The switch table

is initially empty. Show the state of the switch table before and after each of these events. For each of these events, identify the link(s) on which the transmitted frame will be forwarded, and briefly justify your answers.

P27. In this problem, we explore the use of small packets for Voice-over-IP applications. One of the drawbacks of a small packet size is that a large fraction of link bandwidth is consumed by overhead bytes. To this end, suppose that the packet consists of P bytes and 5 bytes of header.

a. Consider sending a digitally encoded voice source directly. Suppose the source is encoded at a constant rate of 128 kbps. Assume each packet is entirely filled before the source sends the packet into the network. The time required to fill a packet is the **packetization delay**. In terms of L, determine the packetization delay in milliseconds.

b. Packetization delays greater than 20 msec can cause a noticeable and unpleasant echo. Determine the packetization delay for $L = 1{,}500$ bytes (roughly corresponding to a maximum-sized Ethernet packet) and for $L = 50$ (corresponding to an ATM packet).

c. Calculate the store-and-forward delay at a single switch for a link rate of $R = 622$ Mbps for $L = 1{,}500$ bytes, and for $L = 50$ bytes.

d. Comment on the advantages of using a small packet size.

P28. Consider the single switch VLAN in Figure 6.25, and assume an external router is connected to switch port 1. Assign IP addresses to the EE and CS hosts and router interface. Trace the steps taken at both the network layer and the link layer to transfer an IP datagram from an EE host to a CS host (*Hint:* Reread the discussion of Figure 6.19 in the text).

P29. Consider the MPLS network shown in Figure 6.29, and suppose that routers R5 and R6 are now MPLS enabled. Suppose that we want to perform traffic engineering so that packets from R6 destined for A are switched to A via R6-R4-R3-R1, and packets from R5 destined for A are switched via R5-R4-R2-R1. Show the MPLS tables in R5 and R6, as well as the modified table in R4, that would make this possible.

P30. Consider again the same scenario as in the previous problem, but suppose that packets from R6 destined for D are switched via R6-R4-R3, while packets from R5 destined to D are switched via R4-R2-R1-R3. Show the MPLS tables in all routers that would make this possible.

P31. In this problem, you will put together much of what you have learned about Internet protocols. Suppose you walk into a room, connect to Ethernet, and want to download a Web page. What are all the protocol steps that take place, starting from powering on your PC to getting the Web page? Assume there is nothing in our DNS or browser caches when you power on your PC. (*Hint:* The steps include the use of Ethernet, DHCP, ARP, DNS, TCP, and

HTTP protocols.) Explicitly indicate in your steps how you obtain the IP and MAC addresses of a gateway router.

P32. Consider the data center network with hierarchical topology in Figure 6.30. Suppose now there are 80 pairs of flows, with ten flows between the first and ninth rack, ten flows between the second and tenth rack, and so on. Further suppose that all links in the network are 10 Gbps, except for the links between hosts and TOR switches, which are 1 Gbps.

a. Each flow has the same data rate; determine the maximum rate of a flow.

b. For the same traffic pattern, determine the maximum rate of a flow for the highly interconnected topology in Figure 6.31.

c. Now suppose there is a similar traffic pattern, but involving 20 hosts on each rack and 160 pairs of flows. Determine the maximum flow rates for the two topologies.

P33. Consider the hierarchical network in Figure 6.30 and suppose that the data center needs to support e-mail and video distribution among other applications. Suppose four racks of servers are reserved for e-mail and four racks are reserved for video. For each of the applications, all four racks must lie below a single tier-2 switch since the tier-2 to tier-1 links do not have sufficient bandwidth to support the intra-application traffic. For the e-mail application, suppose that for 99.9 percent of the time only three racks are used, and that the video application has identical usage patterns.

a. For what fraction of time does the e-mail application need to use a fourth rack? How about for the video application?

b. Assuming e-mail usage and video usage are independent, for what fraction of time do (equivalently, what is the probability that) both applications need their fourth rack?

c. Suppose that it is acceptable for an application to have a shortage of servers for 0.001 percent of time or less (causing rare periods of performance degradation for users). Discuss how the topology in Figure 6.31 can be used so that only seven racks are collectively assigned to the two applications (assuming that the topology can support all the traffic).

Wireshark Labs

At the Companion website for this textbook, http://www.pearsonhighered.com/cs-resources/, you'll find a Wireshark lab that examines the operation of the IEEE 802.3 protocol and the Wireshark frame format. A second Wireshark lab examines packet traces taken in a home network scenario.

Simon S. Lam

Simon S. Lam is Professor and Regents Chair in Computer Sciences at the University of Texas at Austin. From 1971 to 1974, he was with the ARPA Network Measurement Center at UCLA, where he worked on satellite and radio packet switching. He led a research group that invented secure sockets and prototyped, in 1993, the first secure sockets layer named Secure Network Programming, which won the 2004 ACM Software System Award. His research interests are in design and analysis of network protocols and security services. He received his BSEE from Washington State University and his MS and PhD from UCLA. He was elected to the National Academy of Engineering in 2007.

Why did you decide to specialize in networking?

When I arrived at UCLA as a new graduate student in Fall 1969, my intention was to study control theory. Then I took the queuing theory classes of Leonard Kleinrock and was very impressed by him. For a while, I was working on adaptive control of queuing systems as a possible thesis topic. In early 1972, Larry Roberts initiated the ARPAnet Satellite System project (later called Packet Satellite). Professor Kleinrock asked me to join the project. The first thing we did was to introduce a simple, yet realistic, backoff algorithm to the slotted ALOHA protocol. Shortly thereafter, I found many interesting research problems, such as ALOHA's instability problem and need for adaptive backoff, which would form the core of my thesis.

You were active in the early days of the Internet in the 1970s, beginning with your student days at UCLA. What was it like then? Did people have any inkling of what the Internet would become?

The atmosphere was really no different from other system-building projects I have seen in industry and academia. The initially stated goal of the ARPAnet was fairly modest, that is, to provide access to expensive computers from remote locations so that many more scientists could use them. However, with the startup of the Packet Satellite project in 1972 and the Packet Radio project in 1973, ARPA's goal had expanded substantially. By 1973, ARPA was building three different packet networks at the same time, and it became necessary for Vint Cerf and Bob Kahn to develop an interconnection strategy.

Back then, all of these progressive developments in networking were viewed (I believe) as logical rather than magical. No one could have envisioned the scale of the Internet and power of personal computers today. It was a decade before appearance of the first PCs. To put things in perspective, most students submitted their computer programs

as decks of punched cards for batch processing. Only some students had direct access to computers, which were typically housed in a restricted area. Modems were slow and still a rarity. As a graduate student, I had only a phone on my desk, and I used pencil and paper to do most of my work.

Where do you see the field of networking and the Internet heading in the future?

In the past, the simplicity of the Internet's IP protocol was its greatest strength in vanquishing competition and becoming the *de facto* standard for internetworking. Unlike competitors, such as X.25 in the 1980s and ATM in the 1990s, IP can run on top of any link-layer networking technology, because it offers only a best-effort datagram service. Thus, any packet network can connect to the Internet.

Today, IP's greatest strength is actually a shortcoming. IP is like a straitjacket that confines the Internet's development to specific directions. In recent years, many researchers have redirected their efforts to the application layer only. There is also a great deal of research on wireless ad hoc networks, sensor networks, and satellite networks. These networks can be viewed either as stand-alone systems or link-layer systems, which can flourish because they are outside of the IP straitjacket.

Many people are excited about the possibility of P2P systems as a platform for novel Internet applications. However, P2P systems are highly inefficient in their use of Internet resources. A concern of mine is whether the transmission and switching capacity of the Internet core will continue to increase faster than the traffic demand on the Internet as it grows to interconnect all kinds of devices and support future P2P-enabled applications. Without substantial overprovisioning of capacity, ensuring network stability in the presence of malicious attacks and congestion will continue to be a significant challenge.

The Internet's phenomenal growth also requires the allocation of new IP addresses at a rapid rate to network operators and enterprises worldwide. At the current rate, the pool of unallocated IPv4 addresses would be depleted in a few years. When that happens, large contiguous blocks of address space can only be allocated from the IPv6 address space. Since adoption of IPv6 is off to a slow start, due to lack of incentives for early adopters, IPv4 and IPv6 will most likely co-exist on the Internet for many years to come. Successful migration from an IPv4-dominant Internet to an IPv6-dominant Internet will require a substantial global effort.

What is the most challenging part of your job?

The most challenging part of my job as a professor is teaching and motivating *every* student in my class, and *every* doctoral student under my supervision, rather than just the high achievers. The very bright and motivated may require a little guidance but not much else.

I often learn more from these students than they learn from me. Educating and motivating the underachievers present a major challenge.

What impacts do you foresee technology having on learning in the future?

Eventually, almost all human knowledge will be accessible through the Internet, which will be the most powerful tool for learning. This vast knowledge base will have the potential of leveling the playing field for students all over the world. For example, motivated students in any country will be able to access the best-class Web sites, multimedia lectures, and teaching materials. Already, it was said that the IEEE and ACM digital libraries have accelerated the development of computer science researchers in China. In time, the Internet will transcend all geographic barriers to learning.

CHAPTER 7

Wireless and Mobile Networks

In the telephony world, the past 20 years have arguably been the golden years of cellular telephony. The number of worldwide mobile cellular subscribers increased from 34 million in 1993 to nearly 7.0 billion subscribers by 2014, with the number of cellular subscribers now surpassing the number of wired telephone lines. There are now a larger number of mobile phone subscriptions than there are people on our planet. The many advantages of cell phones are evident to all—anywhere, anytime, untethered access to the global telephone network via a highly portable lightweight device. More recently, laptops, smartphones, and tablets are wirelessly connected to the Internet via a cellular or WiFi network. And increasingly, devices such as gaming consoles, thermostats, home security systems, home appliances, watches, eye glasses, cars, traffic control systems and more are being wirelessly connected to the Internet.

From a networking standpoint, the challenges posed by networking these wireless and mobile devices, particularly at the link layer and the network layer, are so different from traditional wired computer networks that an individual chapter devoted to the study of wireless and mobile networks (i.e., *this* chapter) is appropriate.

We'll begin this chapter with a discussion of mobile users, wireless links, and networks, and their relationship to the larger (typically wired) networks to which they connect. We'll draw a distinction between the challenges posed by the *wireless* nature of the communication links in such networks, and by the *mobility* that these wireless links enable. Making this important distinction—between wireless and mobility—will allow us to better isolate, identify, and master the key concepts in each area. Note that there are indeed many networked environments in which the network nodes are wireless but not mobile (e.g., wireless home or office networks with

stationary workstations and large displays), and that there are limited forms of mobility that do not require wireless links (e.g., a worker who uses a wired laptop at home, shuts down the laptop, drives to work, and attaches the laptop to the company's wired network). Of course, many of the most exciting networked environments are those in which users are both wireless *and* mobile—for example, a scenario in which a mobile user (say in the back seat of car) maintains a Voice-over-IP call and multiple ongoing TCP connections while racing down the autobahn at 160 kilometers per hour, soon in an autonomous vehicle. It is here, at the intersection of wireless and mobility, that we'll find the most interesting technical challenges!

We'll begin by illustrating the setting in which we'll consider wireless communication and mobility—a network in which wireless (and possibly mobile) users are connected into the larger network infrastructure by a wireless link at the network's edge. We'll then consider the characteristics of this wireless link in Section 7.2. We include a brief introduction to code division multiple access (CDMA), a shared-medium access protocol that is often used in wireless networks, in Section 7.2. In Section 7.3, we'll examine the link-level aspects of the IEEE 802.11 (WiFi) wireless LAN standard in some depth; we'll also say a few words about Bluetooth and other wireless personal area networks. In Section 7.4, we'll provide an overview of cellular Internet access, including 3G and emerging 4G cellular technologies that provide both voice and high-speed Internet access. In Section 7.5, we'll turn our attention to mobility, focusing on the problems of locating a mobile user, routing to the mobile user, and "handing off" the mobile user who dynamically moves from one point of attachment to the network to another. We'll examine how these mobility services are implemented in the mobile IP standard in enterprise 802.11 networks, and in LTE cellular networks in Sections 7.6 and 7.7, respectively. Finally, we'll consider the impact of wireless links and mobility on transport-layer protocols and networked applications in Section 7.8.

7.1 Introduction

Figure 7.1 shows the setting in which we'll consider the topics of wireless data communication and mobility. We'll begin by keeping our discussion general enough to cover a wide range of networks, including both wireless LANs such as IEEE 802.11 and cellular networks such as a 4G network; we'll drill down into a more detailed discussion of specific wireless architectures in later sections. We can identify the following elements in a wireless network:

- *Wireless hosts.* As in the case of wired networks, hosts are the end-system devices that run applications. A **wireless host** might be a laptop, tablet, smartphone, or desktop computer. The hosts themselves may or may not be mobile.

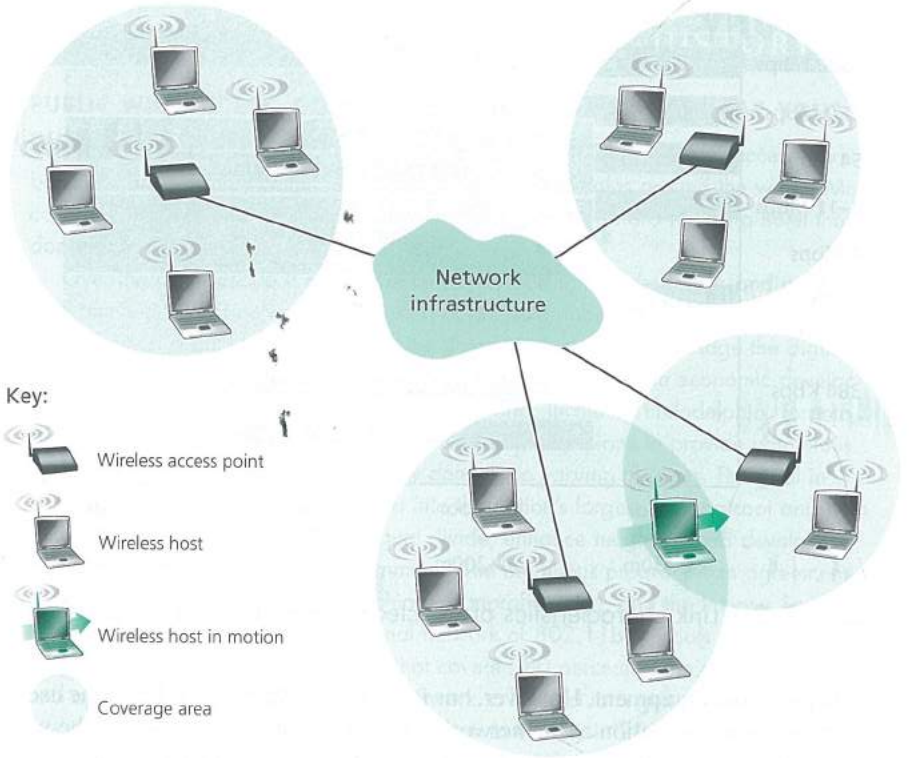


Figure 7.1 ♦ Elements of a wireless network

- *Wireless links.* A host connects to a base station (defined below) or to another wireless host through a **wireless communication link**. Different wireless link technologies have different transmission rates and can transmit over different distances. Figure 7.2 shows two key characteristics (coverage area and link rate) of the more popular wireless network standards. (The figure is only meant to provide a rough idea of these characteristics. For example, some of these types of networks are only now being deployed, and some link rates can increase or decrease beyond the values shown depending on distance, channel conditions, and the number of users in the wireless network.) We'll cover these standards later in the first half of this chapter; we'll also consider other wireless link characteristics (such as their bit error rates and the causes of bit errors) in Section 7.2.

  In Figure 7.1, wireless links connect wireless hosts located at the edge of the network into the larger network infrastructure. We hasten to add that wireless links are also sometimes used *within* a network to connect routers, switches, and

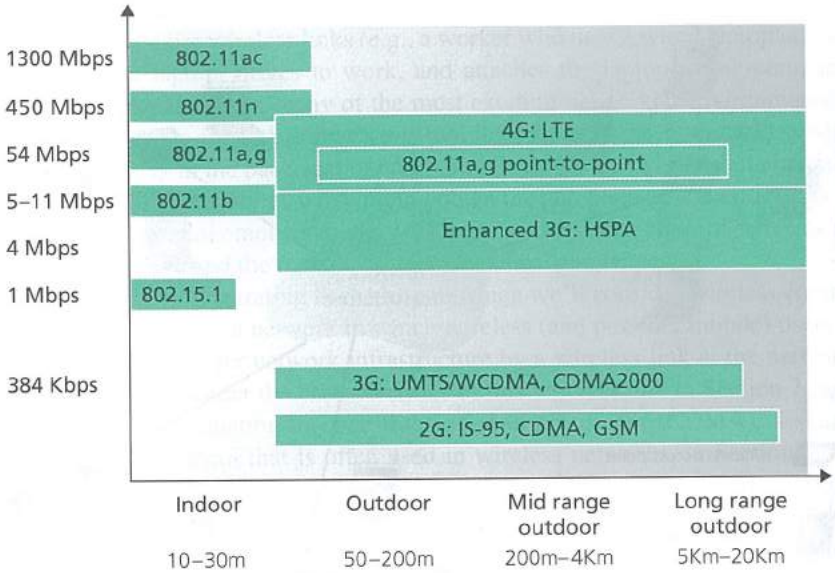


Figure 7.2 ♦ Link characteristics of selected wireless network standards

other network equipment. However, our focus in this chapter will be on the use of wireless communication at the network edge, as it is here that many of the most exciting technical challenges, and most of the growth, are occurring.

- *Base station.* The **base station** is a key part of the wireless network infrastructure. Unlike the wireless host and wireless link, a base station has no obvious counterpart in a wired network. A base station is responsible for sending and receiving data (e.g., packets) to and from a wireless host that is associated with that base station. A base station will often be responsible for coordinating the transmission of multiple wireless hosts with which it is associated. When we say a wireless host is "associated" with a base station, we mean that (1) the host is within the wireless communication distance of the base station, and (2) the host uses that base station to relay data between it (the host) and the larger network. **Cell towers** in cellular networks and **access points** in 802.11 wireless LANs are examples of base stations.

  In Figure 7.1, the base station is connected to the larger network (e.g., the Internet, corporate or home network, or telephone network), thus functioning as a link-layer relay between the wireless host and the rest of the world with which the host communicates.

  Hosts associated with a base station are often referred to as operating in **infrastructure mode**, since all traditional network services (e.g., address assignment and routing) are provided by the network to which a host is connected via

CASE HISTORY

PUBLIC WIFI ACCESS: COMING SOON TO A LAMP POST NEAR YOU?

WiFi hotspots—public locations where users can find 802.11 wireless access—are becoming increasingly common in hotels, airports, and cafés around the world. Most college campuses offer ubiquitous wireless access, and it's hard to find a hotel that doesn't offer wireless Internet access.

Over the past decade a number of cities have designed, deployed, and operated municipal WiFi networks. The vision of providing ubiquitous WiFi access to the community as a public service (much like streetlights)—helping to bridge the digital divide by providing Internet access to all citizens and to promote economic development—is compelling. Many cities around the world, including Philadelphia, Toronto, Hong Kong, Minneapolis, London, and Auckland, have plans to provide ubiquitous wireless within the city, or have already done so to varying degrees. The goal in Philadelphia was to "turn Philadelphia into the nation's largest WiFi hotspot and help to improve education, bridge the digital divide, enhance neighborhood development, and reduce the costs of government." The ambitious program—an agreement between the city, Wireless Philadelphia (a nonprofit entity), and the Internet Service Provider Earthlink—built an operational network of 802.11b hotspots on streetlamp pole arms and traffic control devices that covered 80 percent of the city. But financial and operational concerns caused the network to be sold to a group of private investors in 2008, who later sold the network back to the city in 2010. Other cities, such as Minneapolis, Toronto, Hong Kong, and Auckland, have had success with smaller-scale efforts.

The fact that 802.11 networks operate in the unlicensed spectrum (and hence can be deployed without purchasing expensive spectrum use rights) would seem to make them financially attractive. However, 802.11 access points (see Section 7.3) have much shorter ranges than 4G cellular base stations (see Section 7.4), requiring a larger number of deployed endpoints to cover the same geographic region. Cellular data networks providing Internet access, on the other hand, operate in the licensed spectrum. Cellular providers pay billions of dollars for spectrum access rights for their networks, making cellular data networks a business rather than municipal undertaking.

the base station. In **ad hoc networks**, wireless hosts have no such infrastructure with which to connect. In the absence of such infrastructure, the hosts themselves must provide for services such as routing, address assignment, DNS-like name translation, and more.

When a mobile host moves beyond the range of one base station and into the range of another, it will change its point of attachment into the larger network

(i.e., change the base station with which it is associated)—a process referred to as **handoff**. Such mobility raises many challenging questions. If a host can move, how does one find the mobile host's current location in the network so that data can be forwarded to that mobile host? How is addressing performed, given that a host can be in one of many possible locations? If the host moves *during* a TCP connection or phone call, how is data routed so that the connection continues uninterrupted? These and many (many!) other questions make wireless and mobile networking an area of exciting networking research.

- *Network infrastructure.* This is the larger network with which a wireless host may wish to communicate.

Having discussed the "pieces" of a wireless network, we note that these pieces can be combined in many different ways to form different types of wireless networks. You may find a taxonomy of these types of wireless networks useful as you read on in this chapter, or read/learn more about wireless networks beyond this book. At the highest level we can classify wireless networks according to two criteria: *(i)* whether a packet in the wireless network crosses exactly *one wireless hop or multiple wireless hops*, and *(ii)* whether there is *infrastructure* such as a base station in the network:

- *Single-hop, infrastructure-based.* These networks have a base station that is connected to a larger wired network (e.g., the Internet). Furthermore, all communication is between this base station and a wireless host over a single wireless hop. The 802.11 networks you use in the classroom, café, or library; and the 4G LTE data networks that we will learn about shortly all fall in this category. The vast majority of our daily interactions are with single-hop, infrastructure-based wireless networks.
- *Single-hop, infrastructure-less.* In these networks, there is no base station that is connected to a wireless network. However, as we will see, one of the nodes in this single-hop network may coordinate the transmissions of the other nodes. Bluetooth networks (that connect small wireless devices such as keyboards, speakers, and headsets, and which we will study in Section 7.3.6) and 802.11 networks in ad hoc mode are single-hop, infrastructure-less networks.
- *Multi-hop, infrastructure-based.* In these networks, a base station is present that is wired to the larger network. However, some wireless nodes may have to relay their communication through other wireless nodes in order to communicate via the base station. Some wireless sensor networks and so-called **wireless mesh networks** fall in this category.
- *Multi-hop, infrastructure-less.* There is no base station in these networks, and nodes may have to relay messages among several other nodes in order to reach a destination. Nodes may also be mobile, with connectivity changing among nodes—a class of networks known as **mobile ad hoc networks (MANETs)**.

If the mobile nodes are vehicles, the network is a **vehicular ad hoc network (VANET)**. As you might imagine, the development of protocols for such networks is challenging and is the subject of much ongoing research.

In this chapter, we'll mostly confine ourselves to single-hop networks, and then mostly to infrastructure-based networks.

Let's now dig deeper into the technical challenges that arise in wireless and mobile networks. We'll begin by first considering the individual wireless link, deferring our discussion of mobility until later in this chapter.

7.2 Wireless Links and Network Characteristics

Let's begin by considering a simple wired network, say a home network, with a wired Ethernet switch (see Section 6.4) interconnecting the hosts. If we replace the wired Ethernet with a wireless 802.11 network, a wireless network interface would replace the host's wired Ethernet interface, and an access point would replace the Ethernet switch, but virtually no changes would be needed at the network layer or above. This suggests that we focus our attention on the link layer when looking for important differences between wired and wireless networks. Indeed, we can find a number of important differences between a wired link and a wireless link:

- *Decreasing signal strength.* Electromagnetic radiation attenuates as it passes through matter (e.g., a radio signal passing through a wall). Even in free space, the signal will disperse, resulting in decreased signal strength (sometimes referred to as **path loss**) as the distance between sender and receiver increases.
- *Interference from other sources.* Radio sources transmitting in the same frequency band will interfere with each other. For example, 2.4 GHz wireless phones and 802.11b wireless LANs transmit in the same frequency band. Thus, the 802.11b wireless LAN user talking on a 2.4 GHz wireless phone can expect that neither the network nor the phone will perform particularly well. In addition to interference from transmitting sources, electromagnetic noise within the environment (e.g., a nearby motor, a microwave) can result in interference.
- *Multipath propagation.* **Multipath propagation** occurs when portions of the electromagnetic wave reflect off objects and the ground, taking paths of different lengths between a sender and receiver. This results in the blurring of the received signal at the receiver. Moving objects between the sender and receiver can cause multipath propagation to change over time.

For a detailed discussion of wireless channel characteristics, models, and measurements, see [Anderson 1995].

The discussion above suggests that bit errors will be more common in wireless links than in wired links. For this reason, it is perhaps not surprising that wireless link protocols (such as the 802.11 protocol we'll examine in the following section) employ not only powerful CRC error detection codes, but also link-level reliable-data-transfer protocols that retransmit corrupted frames.

Having considered the impairments that can occur on a wireless channel, let's next turn our attention to the host receiving the wireless signal. This host receives an electromagnetic signal that is a combination of a degraded form of the original signal transmitted by the sender (degraded due to the attenuation and multipath propagation effects that we discussed above, among others) and background noise in the environment. The **signal-to-noise ratio (SNR)** is a relative measure of the strength of the received signal (i.e., the information being transmitted) and this noise. The SNR is typically measured in units of decibels (dB), a unit of measure that some think is used by electrical engineers primarily to confuse computer scientists. The SNR, measured in dB, is twenty times the ratio of the base-10 logarithm of the amplitude of the received signal to the amplitude of the noise. For our purposes here, we need only know that a larger SNR makes it easier for the receiver to extract the transmitted signal from the background noise.

Figure 7.3 (adapted from [Holland 2001]) shows the bit error rate (BER)—roughly speaking, the probability that a transmitted bit is received in error at the receiver—versus the SNR for three different modulation techniques for encoding information for transmission on an idealized wireless channel. The theory of modulation and coding, as well as signal extraction and BER, is well beyond the scope of

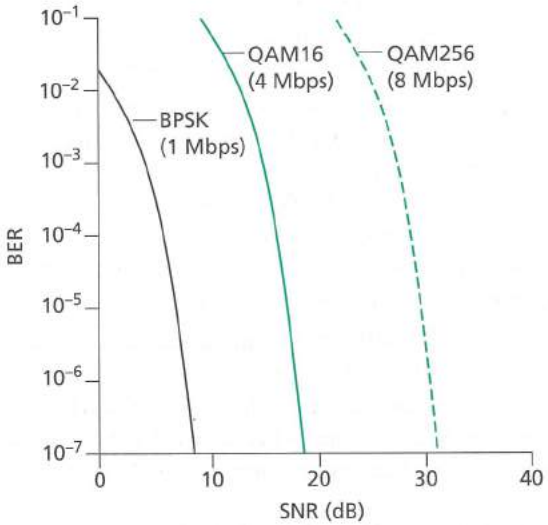


Figure 7.3 ♦ Bit error rate, transmission rate, and SNR

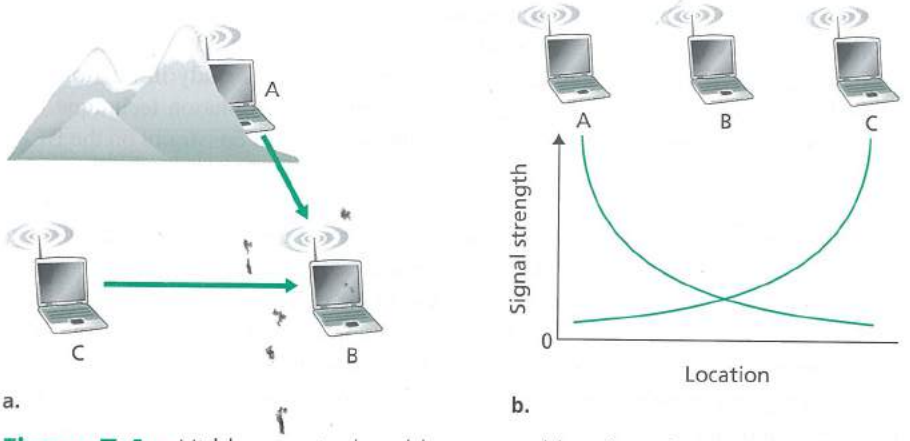


Figure 7.4 ♦ Hidden terminal problem caused by obstacle (a) and fading (b)

this text (see [Schwartz 1980] for a discussion of these topics). Nonetheless, Figure 7.3 illustrates several physical-layer characteristics that are important in understanding higher-layer wireless communication protocols:

- *For a given modulation scheme, the higher the SNR, the lower the BER.* Since a sender can increase the SNR by increasing its transmission power, a sender can decrease the probability that a frame is received in error by increasing its transmission power. Note, however, that there is arguably little practical gain in increasing the power beyond a certain threshold, say to decrease the BER from 10^{-12} to 10^{-13}. There are also *disadvantages* associated with increasing the transmission power: More energy must be expended by the sender (an important concern for battery-powered mobile users), and the sender's transmissions are more likely to interfere with the transmissions of another sender (see Figure 7.4(b)).
- *For a given SNR, a modulation technique with a higher bit transmission rate (whether in error or not) will have a higher BER.* For example, in Figure 7.3, with an SNR of 10 dB, BPSK modulation with a transmission rate of 1 Mbps has a BER of less than 10^{-7}, while with QAM16 modulation with a transmission rate of 4 Mbps, the BER is 10^{-1}, far too high to be practically useful. However, with an SNR of 20 dB, QAM16 modulation has a transmission rate of 4 Mbps and a BER of 10^{-7}, while BPSK modulation has a transmission rate of only 1 Mbps and a BER that is so low as to be (literally) "off the charts." If one can tolerate a BER of 10^{-7}, the higher transmission rate offered by QAM16 would make it the preferred modulation technique in this situation. These considerations give rise to the final characteristic, described next.
- *Dynamic selection of the physical-layer modulation technique can be used to adapt the modulation technique to channel conditions.* The SNR (and hence

the BER) may change as a result of mobility or due to changes in the environment. Adaptive modulation and coding are used in cellular data systems and in the 802.11 WiFi and 4G cellular data networks that we'll study in Sections 7.3 and 7.4. This allows, for example, the selection of a modulation technique that provides the highest transmission rate possible subject to a constraint on the BER, for given channel characteristics.

A higher and time-varying bit error rate is not the only difference between a wired and wireless link. Recall that in the case of wired broadcast links, all nodes receive the transmissions from all other nodes. In the case of wireless links, the situation is not as simple, as shown in Figure 7.4. Suppose that Station A is transmitting to Station B. Suppose also that Station C is transmitting to Station B. With the so-called **hidden terminal problem**, physical obstructions in the environment (for example, a mountain or a building) may prevent A and C from hearing each other's transmissions, even though A's and C's transmissions are indeed interfering at the destination, B. This is shown in Figure 7.4(a). A second scenario that results in undetectable collisions at the receiver results from the **fading** of a signal's strength as it propagates through the wireless medium. Figure 7.4(b) illustrates the case where A and C are placed such that their signals are not strong enough to detect each other's transmissions, yet their signals *are* strong enough to interfere with each other at station B. As we'll see in Section 7.3, the hidden terminal problem and fading make multiple access in a wireless network considerably more complex than in a wired network.

7.2.1 CDMA

Recall from Chapter 6 that when hosts communicate over a shared medium, a protocol is needed so that the signals sent by multiple senders do not interfere at the receivers. In Chapter 6 we described three classes of medium access protocols: channel partitioning, random access, and taking turns. Code division multiple access (CDMA) belongs to the family of channel partitioning protocols. It is prevalent in wireless LAN and cellular technologies. Because CDMA is so important in the wireless world, we'll take a quick look at CDMA now, before getting into specific wireless access technologies in the subsequent sections.

In a CDMA protocol, each bit being sent is encoded by multiplying the bit by a signal (the code) that changes at a much faster rate (known as the **chipping rate**) than the original sequence of data bits. Figure 7.5 shows a simple, idealized CDMA encoding/decoding scenario. Suppose that the rate at which original data bits reach the CDMA encoder defines the unit of time; that is, each original data bit to be transmitted requires a one-bit slot time. Let d_i be the value of the data bit for the ith bit slot. For mathematical convenience, we represent a data bit with a 0 value as -1. Each bit slot is further subdivided into M mini-slots; in Figure 7.5, $M = 8$,

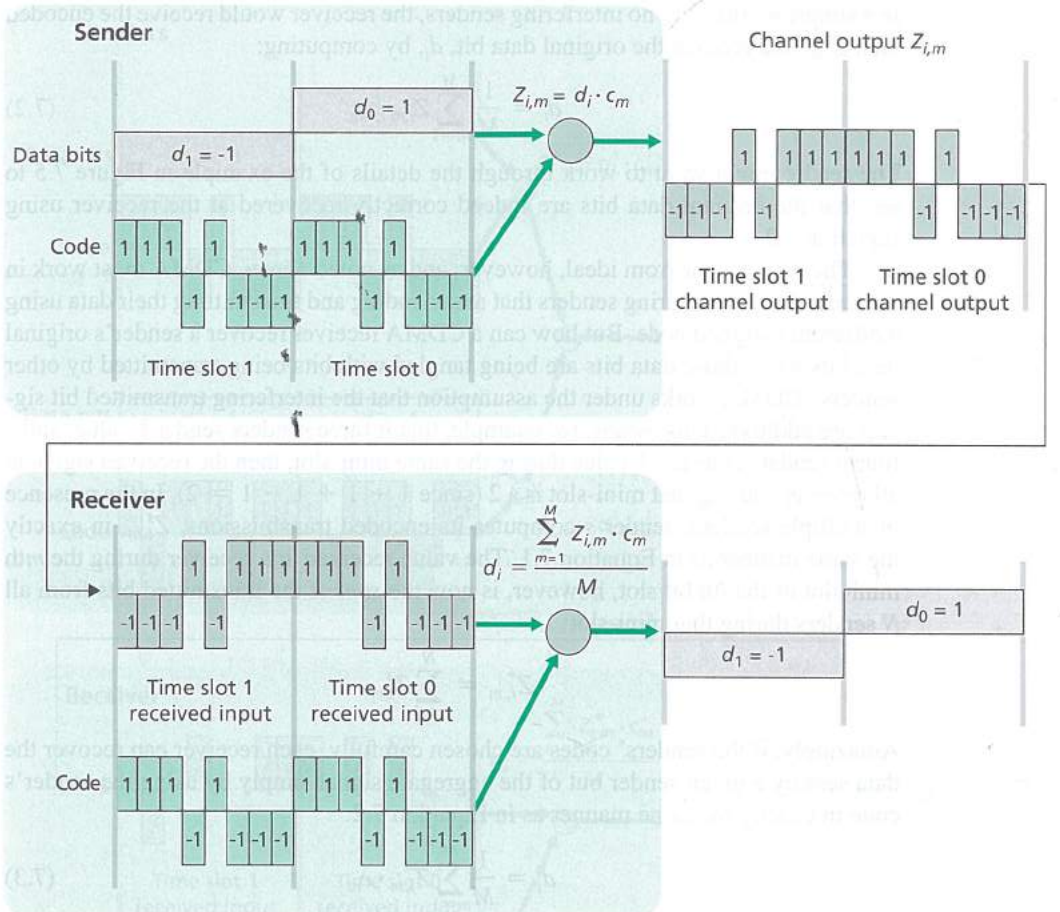


Figure 7.5 ♦ A simple CDMA example: Sender encoding, receiver decoding

although in practice M is much larger. The CDMA code used by the sender consists of a sequence of M values, c_m, $m = 1, \ldots, M$, each taking a $+1$ or -1 value. In the example in Figure 7.5, the M-bit CDMA code being used by the sender is $(1, 1, 1, -1, 1, -1, -1, -1)$.

To illustrate how CDMA works, let us focus on the ith data bit, d_i. For the mth mini-slot of the bit-transmission time of d_i, the output of the CDMA encoder, $Z_{i,m}$, is the value of d_i multiplied by the mth bit in the assigned CDMA code, c_m:

$$Z_{i,m} = d_i \cdot c_m \tag{7.1}$$

In a simple world, with no interfering senders, the receiver would receive the encoded bits, $Z_{i,m}$, and recover the original data bit, d_i, by computing:

$$d_i = \frac{1}{M}\sum_{m=1}^{M} Z_{i,m} \cdot c_m \tag{7.2}$$

The reader might want to work through the details of the example in Figure 7.5 to see that the original data bits are indeed correctly recovered at the receiver using Equation 7.2.

The world is far from ideal, however, and as noted above, CDMA must work in the presence of interfering senders that are encoding and transmitting their data using a different assigned code. But how can a CDMA receiver recover a sender's original data bits when those data bits are being tangled with bits being transmitted by other senders? CDMA works under the assumption that the interfering transmitted bit signals are additive. This means, for example, that if three senders send a 1 value, and a fourth sender sends a -1 value during the same mini-slot, then the received signal at all receivers during that mini-slot is a 2 (since $1 + 1 + 1 - 1 = 2$). In the presence of multiple senders, sender s computes its encoded transmissions, $Z^s_{i,m}$, in exactly the same manner as in Equation 7.1. The value received at a receiver during the mth mini-slot of the ith bit slot, however, is now the *sum* of the transmitted bits from all N senders during that mini-slot:

$$Z^*_{i,m} = \sum_{s=1}^{N} Z^s_{i,m}$$

Amazingly, if the senders' codes are chosen carefully, each receiver can recover the data sent by a given sender out of the aggregate signal simply by using the sender's code in exactly the same manner as in Equation 7.2:

$$d_i = \frac{1}{M}\sum_{m=1}^{M} Z^*_{i,m} \cdot c_m \tag{7.3}$$

as shown in Figure 7.6, for a two-sender CDMA example. The M-bit CDMA code being used by the upper sender is $(1, 1, 1, -1, 1, -1, -1, -1)$, while the CDMA code being used by the lower sender is $(1, -1, 1, 1, 1, -1, 1, 1)$. Figure 7.6 illustrates a receiver recovering the original data bits from the upper sender. Note that the receiver is able to extract the data from sender 1 in spite of the interfering transmission from sender 2.

Recall our cocktail analogy from Chapter 6. A CDMA protocol is similar to having partygoers speaking in multiple languages; in such circumstances humans are actually quite good at locking into the conversation in the language they understand, while filtering out the remaining conversations. We see here that CDMA is a partitioning protocol in that it partitions the codespace (as opposed to time or frequency) and assigns each node a dedicated piece of the codespace.

Our discussion here of CDMA is necessarily brief; in practice a number of difficult issues must be addressed. First, in order for the CDMA receivers to be able

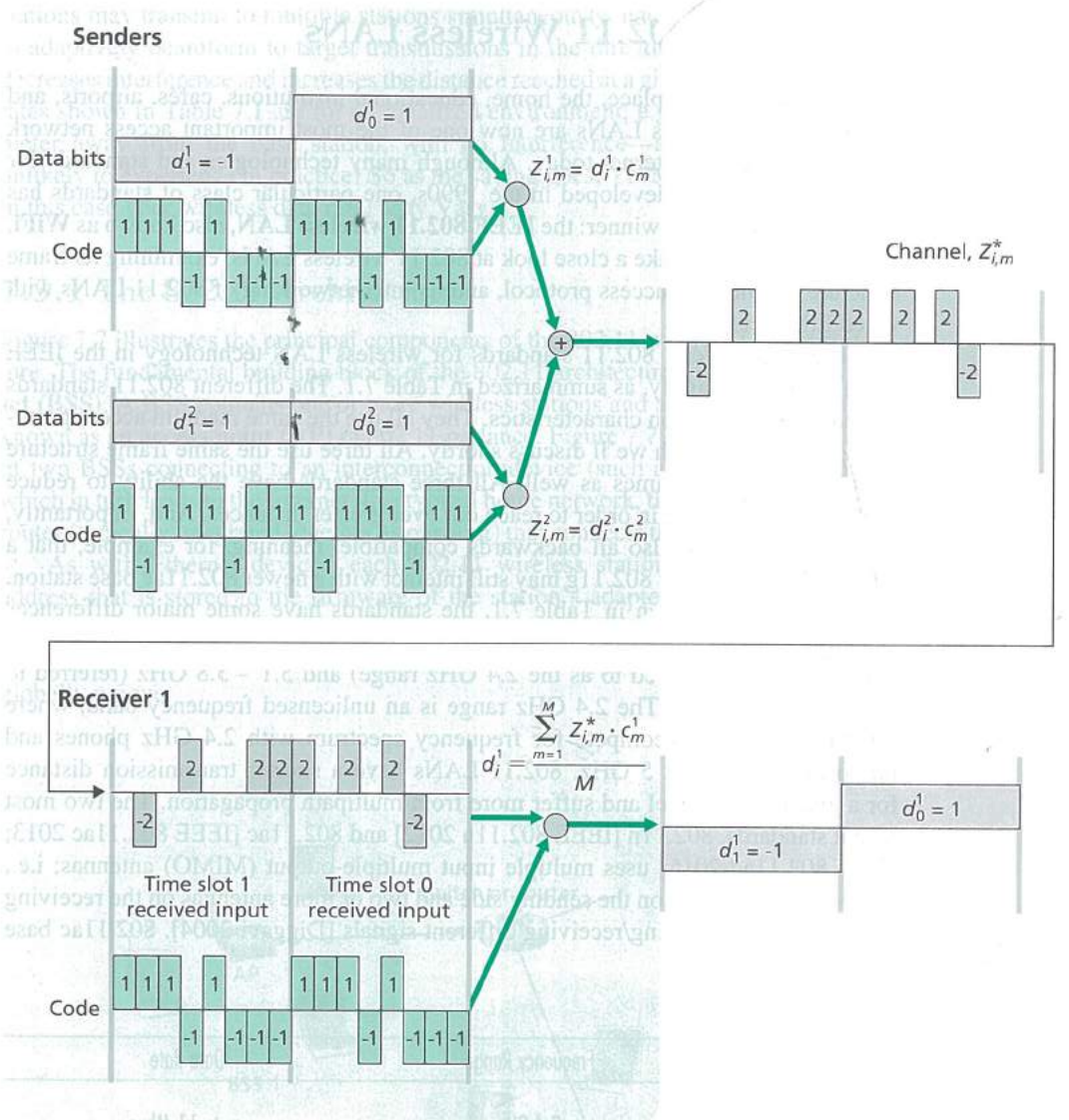


Figure 7.6 ♦ A two-sender CDMA example

to extract a particular sender's signal, the CDMA codes must be carefully chosen. Second, our discussion has assumed that the received signal strengths from various senders are the same; in reality this can be difficult to achieve. There is a considerable body of literature addressing these and other issues related to CDMA; see [Pickholtz 1982; Viterbi 1995] for details.

7.3 WiFi: 802.11 Wireless LANs

Pervasive in the workplace, the home, educational institutions, cafés, airports, and street corners, wireless LANs are now one of the most important access network technologies in the Internet today. Although many technologies and standards for wireless LANs were developed in the 1990s, one particular class of standards has clearly emerged as the winner: the **IEEE 802.11 wireless LAN**, also known as **WiFi**. In this section, we'll take a close look at 802.11 wireless LANs, examining its frame structure, its medium access protocol, and its internetworking of 802.11 LANs with wired Ethernet LANs.

There are several 802.11 standards for wireless LAN technology in the IEEE 802.11 ("WiFi") family, as summarized in Table 7.1. The different 802.11 standards all share some common characteristics. They all use the same medium access protocol, CSMA/CA, which we'll discuss shortly. All three use the same frame structure for their link-layer frames as well. All three standards have the ability to reduce their transmission rate in order to reach out over greater distances. And, importantly, 802.11 products are also all backwards compatible, meaning, for example, that a mobile capable only of 802.11g may still interact with a newer 802.11ac base station.

However, as shown in Table 7.1, the standards have some major differences at the physical layer. 802.11 devices operate in two difference frequency ranges: 2.4–2.485 GHz (referred to as the 2.4 GHz range) and 5.1 – 5.8 GHz (referred to as the 5 GHz range). The 2.4 GHz range is an unlicensed frequency band, where 802.11 devices may compete for frequency spectrum with 2.4 GHz phones and microwave ovens. At 5 GHz, 802.11 LANs have a shorter transmission distance for a given power level and suffer more from multipath propagation. The two most recent standards, 802.11n [IEEE 802.11n 2012] and 802.11ac [IEEE 802.11ac 2013; Cisco 802.11ac 2015] uses multiple input multiple-output (MIMO) antennas; i.e., two or more antennas on the sending side and two or more antennas on the receiving side that are transmitting/receiving different signals [Diggavi 2004]. 802.11ac base

Standard	Frequency Range	Data Rate
802.11b	2.4 GHz	up to 11 Mbps
802.11a	5 GHz	up to 54 Mbps
802.11g	2.4 GHz	up to 54 Mbps
802.11n	2.5 GHz and 5 GHz	up to 450 Mbps
802.11ac	5 GHz	up to 1300 Mbps

Table 7.1 ♦ Summary of IEEE 802.11 standards

stations may transmit to multiple stations simultaneously, and use "smart" antennas to adaptively beamform to target transmissions in the direction of a receiver. This decreases interference and increases the distance reached at a given data rate. The data rates shown in Table 7.1 are for an idealized environment, e.g., a receiver placed 1 meter away from the base station, with no interference—a scenario that we're unlikely to experience in practice! So as the saying goes, YMMV: Your Mileage (or in this case your wireless data rate) May Vary.

7.3.1 The 802.11 Architecture

Figure 7.7 illustrates the principal components of the 802.11 wireless LAN architecture. The fundamental building block of the 802.11 architecture is the **basic service set (BSS)**. A BSS contains one or more wireless stations and a central **base station**, known as an **access point (AP)** in 802.11 parlance. Figure 7.7 shows the AP in each of two BSSs connecting to an interconnection device (such as a switch or router), which in turn leads to the Internet. In a typical home network, there is one AP and one router (typically integrated together as one unit) that connects the BSS to the Internet.

As with Ethernet devices, each 802.11 wireless station has a 6-byte MAC address that is stored in the firmware of the station's adapter (that is, 802.11 network interface card). Each AP also has a MAC address for its wireless interface. As with Ethernet, these MAC addresses are administered by IEEE and are (in theory) globally unique.

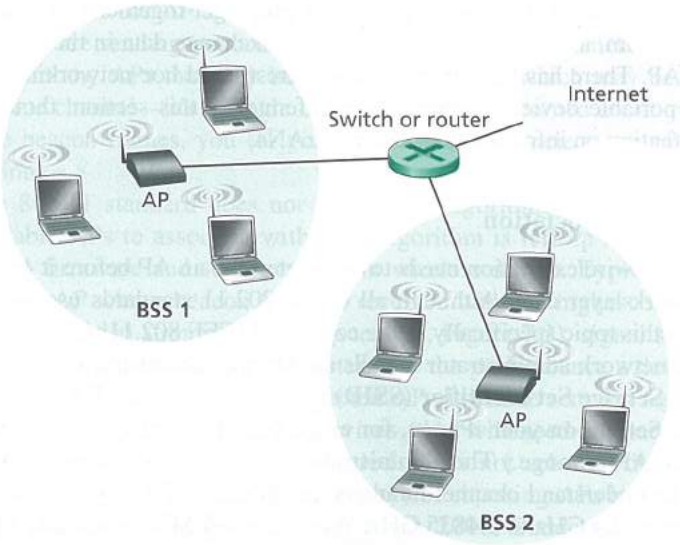


Figure 7.7 ♦ IEEE 802.11 LAN architecture

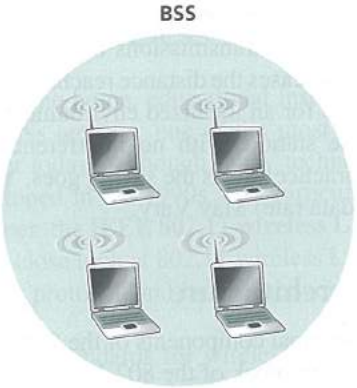


Figure 7.8 ♦ An IEEE 802.11 ad hoc network

As noted in Section 7.1, wireless LANs that deploy APs are often referred to as **infrastructure wireless LANs**, with the "infrastructure" being the APs along with the wired Ethernet infrastructure that interconnects the APs and a router. Figure 7.8 shows that IEEE 802.11 stations can also group themselves together to form an ad hoc network—a network with no central control and with no connections to the "outside world." Here, the network is formed "on the fly," by mobile devices that have found themselves in proximity to each other, that have a need to communicate, and that find no preexisting network infrastructure in their location. An ad hoc network might be formed when people with laptops get together (for example, in a conference room, a train, or a car) and want to exchange data in the absence of a centralized AP. There has been tremendous interest in ad hoc networking, as communicating portable devices continue to proliferate. In this section, though, we'll focus our attention on infrastructure wireless LANs.

Channels and Association

In 802.11, each wireless station needs to associate with an AP before it can send or receive network-layer data. Although all of the 802.11 standards use association, we'll discuss this topic specifically in the context of IEEE 802.11b/g.

When a network administrator installs an AP, the administrator assigns a one- or two-word **Service Set Identifier (SSID)** to the access point. (When you choose Wi-Fi under Setting on your iPhone, for example, a list is displayed showing the SSID of each AP in range.) The administrator must also assign a channel number to the AP. To understand channel numbers, recall that 802.11 operates in the frequency range of 2.4 GHz to 2.4835 GHz. Within this 85 MHz band, 802.11 defines 11 partially overlapping channels. Any two channels are non-overlapping if and only if they are separated by four or more channels. In particular, the set of channels

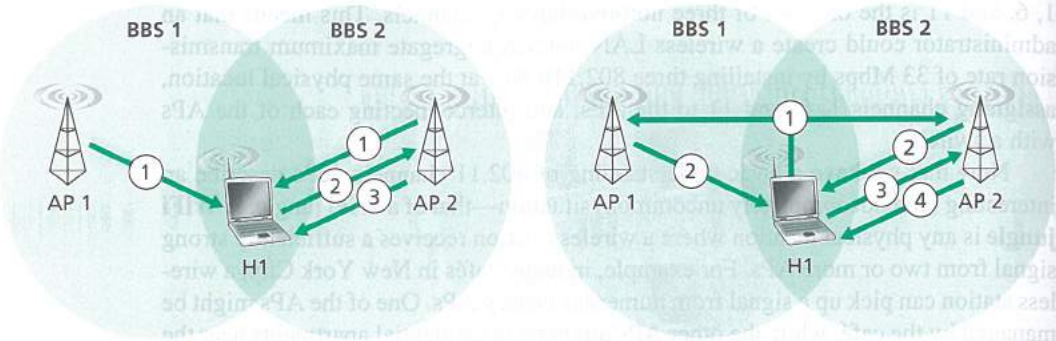
1, 6, and 11 is the only set of three non-overlapping channels. This means that an administrator could create a wireless LAN with an aggregate maximum transmission rate of 33 Mbps by installing three 802.11b APs at the same physical location, assigning channels 1, 6, and 11 to the APs, and interconnecting each of the APs with a switch.

Now that we have a basic understanding of 802.11 channels, let's describe an interesting (and not completely uncommon) situation—that of a WiFi jungle. A **WiFi jungle** is any physical location where a wireless station receives a sufficiently strong signal from two or more APs. For example, in many cafés in New York City, a wireless station can pick up a signal from numerous nearby APs. One of the APs might be managed by the café, while the other APs might be in residential apartments near the café. Each of these APs would likely be located in a different IP subnet and would have been independently assigned a channel.

Now suppose you enter such a WiFi jungle with your phone, tablet, or laptop, seeking wireless Internet access and a blueberry muffin. Suppose there are five APs in the WiFi jungle. To gain Internet access, your wireless device needs to join exactly one of the subnets and hence needs to **associate** with exactly one of the APs. Associating means the wireless device creates a virtual wire between itself and the AP. Specifically, only the associated AP will send data frames (that is, frames containing data, such as a datagram) to your wireless device, and your wireless device will send data frames into the Internet only through the associated AP. But how does your wireless device associate with a particular AP? And more fundamentally, how does your wireless device know which APs, if any, are out there in the jungle?

The 802.11 standard requires that an AP periodically send **beacon frames**, each of which includes the AP's SSID and MAC address. Your wireless device, knowing that APs are sending out beacon frames, scans the 11 channels, seeking beacon frames from any APs that may be out there (some of which may be transmitting on the same channel—it's a jungle out there!). Having learned about available APs from the beacon frames, you (or your wireless device) select one of the APs for association.

The 802.11 standard does not specify an algorithm for selecting which of the available APs to associate with; that algorithm is left up to the designers of the 802.11 firmware and software in your wireless device. Typically, the device chooses the AP whose beacon frame is received with the highest signal strength. While a high signal strength is good (see, e.g., Figure 7.3), signal strength is not the only AP characteristic that will determine the performance a device receives. In particular, it's possible that the selected AP may have a strong signal, but may be overloaded with other affiliated devices (that will need to share the wireless bandwidth at that AP), while an unloaded AP is not selected due to a slightly weaker signal. A number of alternative ways of choosing APs have thus recently been proposed [Vasudevan 2005; Nicholson 2006; Sundaresan 2006]. For an interesting and down-to-earth discussion of how signal strength is measured, see [Bardwell 2004].



a. Passive scanning
1. Beacon frames sent from APs
2. Association Request frame sent: H1 to selected AP
3. Association Response frame sent: Selected AP to H1

a. Active scanning
1. Probe Request frame broadcast from H1
2. Probes Response frame sent from APs
3. Association Request frame sent: H1 to selected AP
4. Association Response frame sent: Selected AP to H1

Figure 7.9 ♦ Active and passive scanning for access points

The process of scanning channels and listening for beacon frames is known as **passive scanning** (see Figure 7.9a). A wireless device can also perform **active scanning**, by broadcasting a probe frame that will be received by all APs within the wireless device's range, as shown in Figure 7.9b. APs respond to the probe request frame with a probe response frame. The wireless device can then choose the AP with which to associate from among the responding APs.

After selecting the AP with which to associate, the wireless device sends an association request frame to the AP, and the AP responds with an association response frame. Note that this second request/response handshake is needed with active scanning, since an AP responding to the initial probe request frame doesn't know which of the (possibly many) responding APs the device will choose to associate with, in much the same way that a DHCP client can choose from among multiple DHCP servers (see Figure 4.21). Once associated with an AP, the device will want to join the subnet (in the IP addressing sense of Section 4.3.3) to which the AP belongs. Thus, the device will typically send a DHCP discovery message (see Figure 4.21) into the subnet via the AP in order to obtain an IP address on the subnet. Once the address is obtained, the rest of the world then views that device simply as another host with an IP address in that subnet.

In order to create an association with a particular AP, the wireless device may be required to authenticate itself to the AP. 802.11 wireless LANs provide a number of alternatives for authentication and access. One approach, used by many companies, is to permit access to a wireless network based on a device's MAC address. A second approach, used by many Internet cafés, employs usernames and passwords.

In both cases, the AP typically communicates with an authentication server, relaying information between the wireless device and the authentication server using a protocol such as RADIUS [RFC 2865] or DIAMETER [RFC 3588]. Separating the authentication server from the AP allows one authentication server to serve many APs, centralizing the (often sensitive) decisions of authentication and access within the single server, and keeping AP costs and complexity low. We'll see in chapter 8 that the new IEEE 802.11i protocol defining security aspects of the 802.11 protocol family takes precisely this approach.

7.3.2 The 802.11 MAC Protocol

Once a wireless device is associated with an AP, it can start sending and receiving data frames to and from the access point. But because multiple wireless devices, or the AP itself may want to transmit data frames at the same time over the same channel, a multiple access protocol is needed to coordinate the transmissions. In the following, we'll refer to the devices or the AP as wireless "stations" that share the multiple access channel. As discussed in Chapter 6 and Section 7.2.1, broadly speaking there are three classes of multiple access protocols: channel partitioning (including CDMA), random access, and taking turns. Inspired by the huge success of Ethernet and its random access protocol, the designers of 802.11 chose a random access protocol for 802.11 wireless LANs. This random access protocol is referred to as **CSMA with collision avoidance**, or more succinctly as **CSMA/CA**. As with Ethernet's CSMA/CD, the "CSMA" in CSMA/CA stands for "carrier sense multiple access," meaning that each station senses the channel before transmitting, and refrains from transmitting when the channel is sensed busy. Although both Ethernet and 802.11 use carrier-sensing random access, the two MAC protocols have important differences. First, instead of using collision detection, 802.11 uses collision-avoidance techniques. Second, because of the relatively high bit error rates of wireless channels, 802.11 (unlike Ethernet) uses a link-layer acknowledgment/retransmission (ARQ) scheme. We'll describe 802.11's collision-avoidance and link-layer acknowledgment schemes below.

Recall from Sections 6.3.2 and 6.4.2 that with Ethernet's collision-detection algorithm, an Ethernet station listens to the channel as it transmits. If, while transmitting, it detects that another station is also transmitting, it aborts its transmission and tries to transmit again after waiting a small, random amount of time. Unlike the 802.3 Ethernet protocol, the 802.11 MAC protocol does *not* implement collision detection. There are two important reasons for this:

- The ability to detect collisions requires the ability to send (the station's own signal) and receive (to determine whether another station is also transmitting) at the same time. Because the strength of the received signal is typically very small compared to the strength of the transmitted signal at the 802.11 adapter, it is costly to build hardware that can detect a collision.

- More importantly, even if the adapter could transmit and listen at the same time (and presumably abort transmission when it senses a busy channel), the adapter would still not be able to detect all collisions, due to the hidden terminal problem and fading, as discussed in Section 7.2.

Because 802.11wireless LANs do not use collision detection, once a station begins to transmit a frame, *it transmits the frame in its entirety*; that is, once a station gets started, there is no turning back. As one might expect, transmitting entire frames (particularly long frames) when collisions are prevalent can significantly degrade a multiple access protocol's performance. In order to reduce the likelihood of collisions, 802.11 employs several collision-avoidance techniques, which we'll shortly discuss.

Before considering collision avoidance, however, we'll first need to examine 802.11's **link-layer acknowledgment** scheme. Recall from Section 7.2 that when a station in a wireless LAN sends a frame, the frame may not reach the destination station intact for a variety of reasons. To deal with this non-negligible chance of failure, the 802.11 MAC protocol uses link-layer acknowledgments. As shown in Figure 7.10, when the destination station receives a frame that passes the CRC, it waits a short period of time known as the **Short Inter-frame Spacing (SIFS)** and then sends back

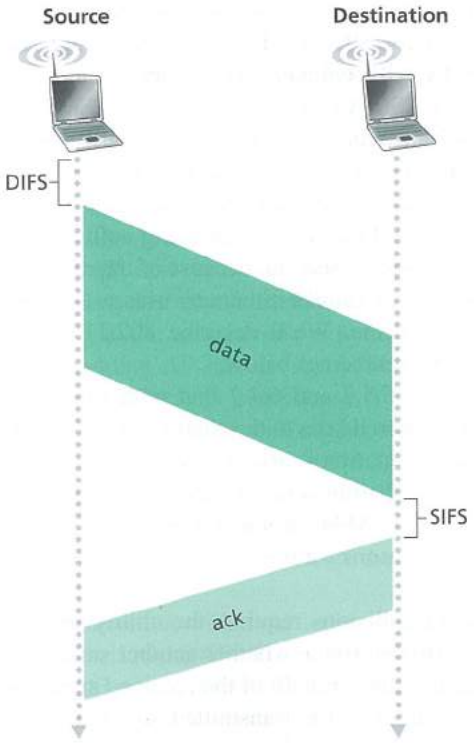


Figure 7.10 ♦ 802.11 uses link-layer acknowledgments

an acknowledgment frame. If the transmitting station does not receive an acknowledgment within a given amount of time, it assumes that an error has occurred and retransmits the frame, using the CSMA/CA protocol to access the channel. If an acknowledgment is not received after some fixed number of retransmissions, the transmitting station gives up and discards the frame.

Having discussed how 802.11 uses link-layer acknowledgments, we're now in a position to describe the 802.11 CSMA/CA protocol. Suppose that a station (wireless device or an AP) has a frame to transmit.

1. If initially the station senses the channel idle, it transmits its frame after a short period of time known as the **Distributed Inter-frame Space (DIFS)**; see Figure 7.10.
2. Otherwise, the station chooses a random backoff value using binary exponential backoff (as we encountered in Section 6.3.2) and counts down this value after DIFS when the channel is sensed idle. While the channel is sensed busy, the counter value remains frozen.
3. When the counter reaches zero (note that this can only occur while the channel is sensed idle), the station transmits the entire frame and then waits for an acknowledgment.
4. If an acknowledgment is received, the transmitting station knows that its frame has been correctly received at the destination station. If the station has another frame to send, it begins the CSMA/CA protocol at step 2. If the acknowledgment isn't received, the transmitting station reenters the backoff phase in step 2, with the random value chosen from a larger interval.

Recall that under Ethernet's CSMA/CD, multiple access protocol (Section 6.3.2), a station begins transmitting as soon as the channel is sensed idle. With CSMA/CA, however, the station refrains from transmitting while counting down, even when it senses the channel to be idle. Why do CSMA/CD and CDMA/CA take such different approaches here?

To answer this question, let's consider a scenario in which two stations each have a data frame to transmit, but neither station transmits immediately because each senses that a third station is already transmitting. With Ethernet's CSMA/CD, the two stations would each transmit as soon as they detect that the third station has finished transmitting. This would cause a collision, which isn't a serious issue in CSMA/CD, since both stations would abort their transmissions and thus avoid the useless transmissions of the remainders of their frames. In 802.11, however, the situation is quite different. Because 802.11 does not detect a collision and abort transmission, a frame suffering a collision will be transmitted in its entirety. The goal in 802.11 is thus to avoid collisions whenever possible. In 802.11, if the two stations sense the channel busy, they both immediately enter random backoff, hopefully choosing different backoff values. If these values are indeed different, once the channel becomes idle, one of the two stations will begin transmitting before the other, and (if the two stations are not hidden from each other) the "losing station" will hear the

"winning station's" signal, freeze its counter, and refrain from transmitting until the winning station has completed its transmission. In this manner, a costly collision is avoided. Of course, collisions can still occur with 802.11 in this scenario: The two stations could be hidden from each other, or the two stations could choose random backoff values that are close enough that the transmission from the station starting first have yet to reach the second station. Recall that we encountered this problem earlier in our discussion of random access algorithms in the context of Figure 6.12.

Dealing with Hidden Terminals: RTS and CTS

The 802.11 MAC protocol also includes a nifty (but optional) reservation scheme that helps avoid collisions even in the presence of hidden terminals. Let's investigate this scheme in the context of Figure 7.11, which shows two wireless stations and one access point. Both of the wireless stations are within range of the AP (whose coverage is shown as a shaded circle) and both have associated with the AP. However, due to fading, the signal ranges of wireless stations are limited to the interiors of the shaded circles shown in Figure 7.11. Thus, each of the wireless stations is hidden from the other, although neither is hidden from the AP.

Let's now consider why hidden terminals can be problematic. Suppose Station H1 is transmitting a frame and halfway through H1's transmission, Station H2 wants to send a frame to the AP. H2, not hearing the transmission from H1, will first wait a DIFS interval and then transmit the frame, resulting in a collision. The channel will therefore be wasted during the entire period of H1's transmission as well as during H2's transmission.

In order to avoid this problem, the IEEE 802.11 protocol allows a station to use a short **Request to Send (RTS)** control frame and a short **Clear to Send (CTS)** control frame to *reserve* access to the channel. When a sender wants to send a DATA

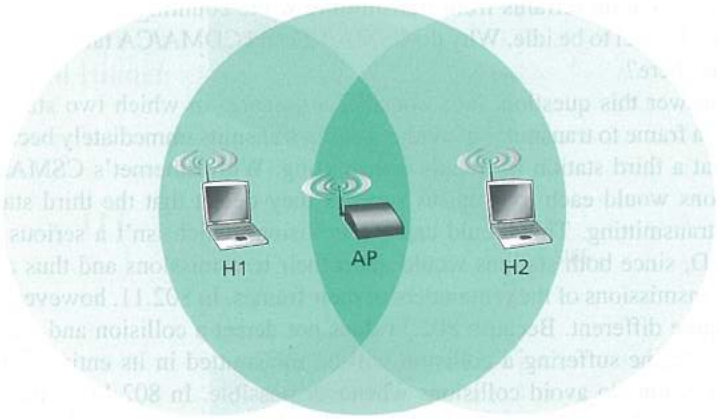


Figure 7.11 ♦ Hidden terminal example: H1 is hidden from H2, and vice versa

frame, it can first send an RTS frame to the AP, indicating the total time required to transmit the DATA frame and the acknowledgment (ACK) frame. When the AP receives the RTS frame, it responds by broadcasting a CTS frame. This CTS frame serves two purposes: It gives the sender explicit permission to send and also instructs the other stations not to send for the reserved duration.

Thus, in Figure 7.12, before transmitting a DATA frame, H1 first broadcasts an RTS frame, which is heard by all stations in its circle, including the AP. The AP then responds

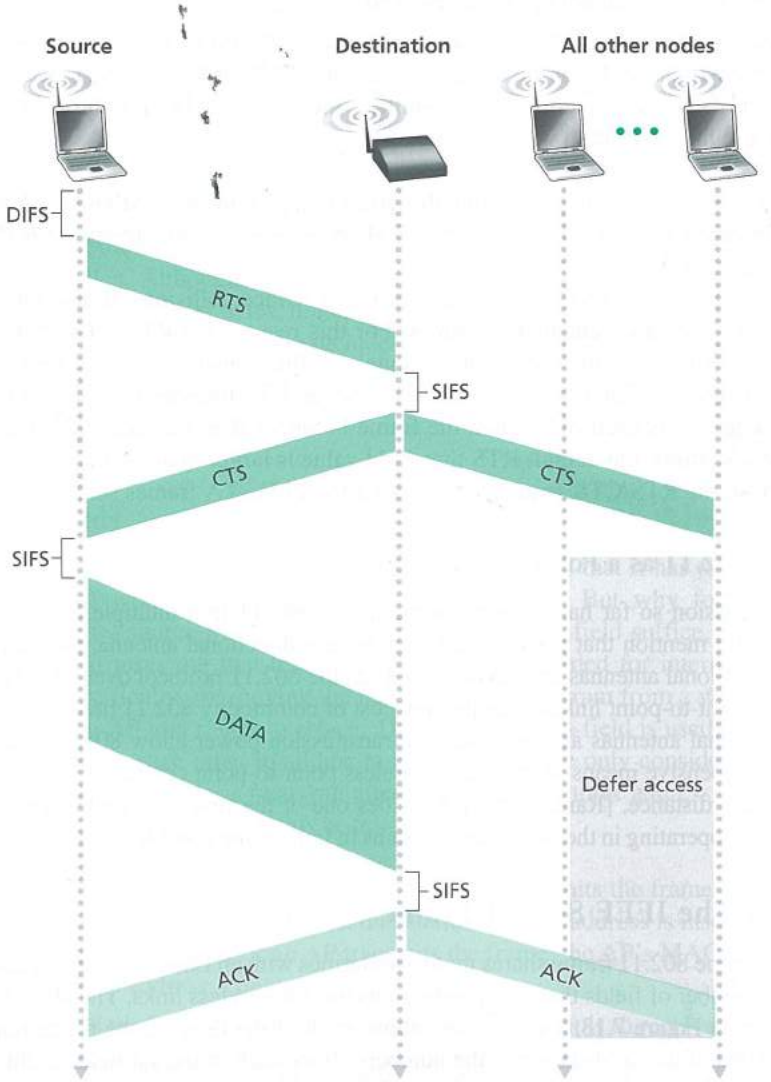


Figure 7.12 ♦ Collision avoidance using the RTS and CTS frames

with a CTS frame, which is heard by all stations within its range, including H1 and H2. Station H2, having heard the CTS, refrains from transmitting for the time specified in the CTS frame. The RTS, CTS, DATA, and ACK frames are shown in Figure 7.12.

The use of the RTS and CTS frames can improve performance in two important ways:

- The hidden station problem is mitigated, since a long DATA frame is transmitted only after the channel has been reserved.
- Because the RTS and CTS frames are short, a collision involving an RTS or CTS frame will last only for the duration of the short RTS or CTS frame. Once the RTS and CTS frames are correctly transmitted, the following DATA and ACK frames should be transmitted without collisions.

You are encouraged to check out the 802.11 applet in the textbook's Web site. This interactive applet illustrates the CSMA/CA protocol, including the RTS/CTS exchange sequence.

Although the RTS/CTS exchange can help reduce collisions, it also introduces delay and consumes channel resources. For this reason, the RTS/CTS exchange is only used (if at all) to reserve the channel for the transmission of a long DATA frame. In practice, each wireless station can set an RTS threshold such that the RTS/CTS sequence is used only when the frame is longer than the threshold. For many wireless stations, the default RTS threshold value is larger than the maximum frame length, so the RTS/CTS sequence is skipped for all DATA frames sent.

Using 802.11 as a Point-to-Point Link

Our discussion so far has focused on the use of 802.11 in a multiple access setting. We should mention that if two nodes each have a directional antenna, they can point their directional antennas at each other and run the 802.11 protocol over what is essentially a point-to-point link. Given the low cost of commodity 802.11 hardware, the use of directional antennas and an increased transmission power allow 802.11 to be used as an inexpensive means of providing wireless point-to-point connections over tens of kilometers distance. [Raman 2007] describes one of the first such multi-hop wireless networks, operating in the rural Ganges plains in India using point-to-point 802.11 links.

7.3.3 The IEEE 802.11 Frame

Although the 802.11 frame shares many similarities with an Ethernet frame, it also contains a number of fields that are specific to its use for wireless links. The 802.11 frame is shown in Figure 7.13. The numbers above each of the fields in the frame represent the lengths of the fields in *bytes*; the numbers above each of the subfields in the frame control field represent the lengths of the subfields in *bits*. Let's now examine the fields in the frame as well as some of the more important subfields in the frame's control field.

Frame (numbers indicate field length in bytes):

2	2	6	6	6	2	6	0-2312	4
Frame control	Duration	Address 1	Address 2	Address 3	Seq control	Address 4	Payload	CRC

Frame control field expanded (numbers indicate field length in bits):

2	2	4	1	1	1	1	1	1	1	1
Protocol version	Type	Subtype	To AP	From AP	More frag	Retry	Power mgt	More data	WEP	Rsvd

Figure 7.13 ♦ The 802.11 frame

Payload and CRC Fields

At the heart of the frame is the payload, which typically consists of an IP datagram or an ARP packet. Although the field is permitted to be as long as 2,312 bytes, it is typically fewer than 1,500 bytes, holding an IP datagram or an ARP packet. As with an Ethernet frame, an 802.11 frame includes a 32-bit cyclic redundancy check (CRC) so that the receiver can detect bit errors in the received frame. As we've seen, bit errors are much more common in wireless LANs than in wired LANs, so the CRC is even more useful here.

Address Fields

Perhaps the most striking difference in the 802.11 frame is that it has *four* address fields, each of which can hold a 6-byte MAC address. But why four address fields? Doesn't a source MAC field and destination MAC field suffice, as they do for Ethernet? It turns out that three address fields are needed for internetworking purposes—specifically, for moving the network-layer datagram from a wireless station through an AP to a router interface. The fourth address field is used when APs forward frames to each other in ad hoc mode. Since we are only considering infrastructure networks here, let's focus our attention on the first three address fields. The 802.11 standard defines these fields as follows:

- Address 2 is the MAC address of the station that transmits the frame. Thus, if a wireless station transmits the frame, that station's MAC address is inserted in the address 2 field. Similarly, if an AP transmits the frame, the AP's MAC address is inserted in the address 2 field.
- Address 1 is the MAC address of the wireless station that is to receive the frame. Thus if a mobile wireless station transmits the frame, address 1 contains the MAC address of the destination AP. Similarly, if an AP transmits the frame, address 1 contains the MAC address of the destination wireless station.

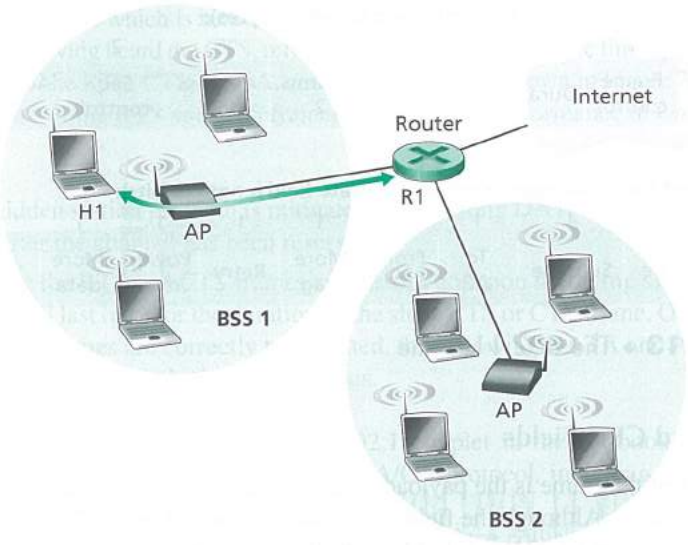


Figure 7.14 ♦ The use of address fields in 802.11 frames: Sending frames between H1 and R1

- To understand address 3, recall that the BSS (consisting of the AP and wireless stations) is part of a subnet, and that this subnet connects to other subnets via some router interface. Address 3 contains the MAC address of this router interface.

To gain further insight into the purpose of address 3, let's walk through an internetworking example in the context of Figure 7.14. In this figure, there are two APs, each of which is responsible for a number of wireless stations. Each of the APs has a direct connection to a router, which in turn connects to the global Internet. We should keep in mind that an AP is a link-layer device, and thus neither "speaks" IP nor understands IP addresses. Consider now moving a datagram from the router interface R1 to the wireless Station H1. The router is not aware that there is an AP between it and H1; from the router's perspective, H1 is just a host in one of the subnets to which it (the router) is connected.

- The router, which knows the IP address of H1 (from the destination address of the datagram), uses ARP to determine the MAC address of H1, just as in an ordinary Ethernet LAN. After obtaining H1's MAC address, router interface R1 encapsulates the datagram within an Ethernet frame. The source address field of this frame contains R1's MAC address, and the destination address field contains H1's MAC address.

- When the Ethernet frame arrives at the AP, the AP converts the 802.3 Ethernet frame to an 802.11 frame before transmitting the frame into the wireless channel. The AP fills in address 1 and address 2 with H1's MAC address and its own MAC address, respectively, as described above. For address 3, the AP inserts the MAC address of R1. In this manner, H1 can determine (from address 3) the MAC address of the router interface that sent the datagram into the subnet.

Now consider what happens when the wireless station H1 responds by moving a datagram from H1 to R1.

- H1 creates an 802.11 frame, filling the fields for address 1 and address 2 with the AP's MAC address and H1's MAC address, respectively, as described above. For address 3, H1 inserts R1's MAC address.
- When the AP receives the 802.11 frame, it converts the frame to an Ethernet frame. The source address field for this frame is H1's MAC address, and the destination address field is R1's MAC address. Thus, address 3 allows the AP to determine the appropriate destination MAC address when constructing the Ethernet frame.

In summary, address 3 plays a crucial role for internetworking the BSS with a wired LAN.

Sequence Number, Duration, and Frame Control Fields

Recall that in 802.11, whenever a station correctly receives a frame from another station, it sends back an acknowledgment. Because acknowledgments can get lost, the sending station may send multiple copies of a given frame. As we saw in our discussion of the rdt2.1 protocol (Section 3.4.1), the use of sequence numbers allows the receiver to distinguish between a newly transmitted frame and the retransmission of a previous frame. The sequence number field in the 802.11 frame thus serves exactly the same purpose here at the link layer as it did in the transport layer in Chapter 3.

Recall that the 802.11 protocol allows a transmitting station to reserve the channel for a period of time that includes the time to transmit its data frame and the time to transmit an acknowledgment. This duration value is included in the frame's duration field (both for data frames and for the RTS and CTS frames).

As shown in Figure 7.13, the frame control field includes many subfields. We'll say just a few words about some of the more important subfields; for a more complete discussion, you are encouraged to consult the 802.11 specification [Held 2001; Crow 1997; IEEE 802.11 1999]. The *type* and *subtype* fields are used to distinguish the association, RTS, CTS, ACK, and data frames. The *to* and *from* fields are used to define the meanings of the different address fields. (These meanings change depending on whether ad hoc or infrastructure modes are used and, in the case of infrastructure mode, whether a wireless station or an AP is sending the frame.) Finally the WEP field indicates whether encryption is being used or not (WEP is discussed in Chapter 8).

7.3.4 Mobility in the Same IP Subnet

In order to increase the physical range of a wireless LAN, companies and universities will often deploy multiple BSSs within the same IP subnet. This naturally raises the issue of mobility among the BSSs—how do wireless stations seamlessly move from one BSS to another while maintaining ongoing TCP sessions? As we'll see in this subsection, mobility can be handled in a relatively straightforward manner when the BSSs are part of the subnet. When stations move between subnets, more sophisticated mobility management protocols will be needed, such as those we'll study in Sections 7.5 and 7.6.

Let's now look at a specific example of mobility between BSSs in the same subnet. Figure 7.15 shows two interconnected BSSs with a host, H1, moving from BSS1 to BSS2. Because in this example the interconnection device that connects the two BSSs is *not* a router, all of the stations in the two BSSs, including the APs, belong to the same IP subnet. Thus, when H1 moves from BSS1 to BSS2, it may keep its IP address and all of its ongoing TCP connections. If the interconnection device were a router, then H1 would have to obtain a new IP address in the subnet in which it was moving. This address change would disrupt (and eventually terminate) any on-going TCP connections at H1. In Section 7.6, we'll see how a network-layer mobility protocol, such as mobile IP, can be used to avoid this problem.

But what specifically happens when H1 moves from BSS1 to BSS2? As H1 wanders away from AP1, H1 detects a weakening signal from AP1 and starts to scan for a stronger signal. H1 receives beacon frames from AP2 (which in many corporate and university settings will have the same SSID as AP1). H1 then disassociates with AP1 and associates with AP2, while keeping its IP address and maintaining its ongoing TCP sessions.

This addresses the handoff problem from the host and AP viewpoint. But what about the switch in Figure 7.15? How does it know that the host has moved from one AP to another? As you may recall from Chapter 6, switches are "self-learning" and automatically build their forwarding tables. This self-learning feature nicely handles

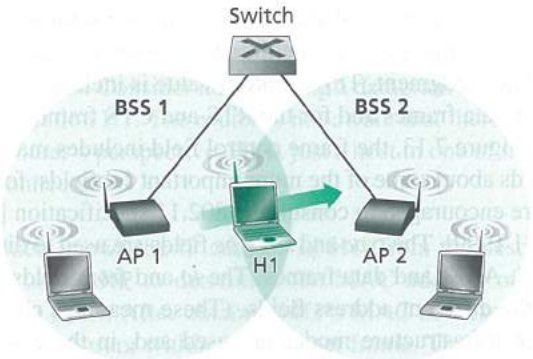


Figure 7.15 ♦ Mobility in the same subnet

occasional moves (for example, when an employee gets transferred from one department to another); however, switches were not designed to support highly mobile users who want to maintain TCP connections while moving between BSSs. To appreciate the problem here, recall that before the move, the switch has an entry in its forwarding table that pairs H1's MAC address with the outgoing switch interface through which H1 can be reached. If H1 is initially in BSS1, then a datagram destined to H1 will be directed to H1 via AP1. Once H1 associates with BSS2, however, its frames should be directed to AP2. One solution (a bit of a hack, really) is for AP2 to send a broadcast Ethernet frame with H1's source address to the switch just after the new association. When the switch receives the frame, it updates its forwarding table, allowing H1 to be reached via AP2. The 802.11f standards group is developing an inter-AP protocol to handle these and related issues.

Our discussion above has focused on mobility with the same LAN subnet. Recall that VLANs, which we studied in Section 6.4.4, can be used to connect together islands of LANs into a large virtual LAN that can span a large geographical region. Mobility among base stations within such a VLAN can be handled in exactly the same manner as above [Yu 2011].

7.3.5 Advanced Features in 802.11

We'll wrap up our coverage of 802.11 with a short discussion of two advanced capabilities found in 802.11 networks. As we'll see, these capabilities are *not* completely specified in the 802.11 standard, but rather are made possible by mechanisms specified in the standard. This allows different vendors to implement these capabilities using their own (proprietary) approaches, presumably giving them an edge over the competition.

802.11 Rate Adaptation

We saw earlier in Figure 7.3 that different modulation techniques (with the different transmission rates that they provide) are appropriate for different SNR scenarios. Consider for example a mobile 802.11 user who is initially 20 meters away from the base station, with a high signal-to-noise ratio. Given the high SNR, the user can communicate with the base station using a physical-layer modulation technique that provides high transmission rates while maintaining a low BER. This is one happy user! Suppose now that the user becomes mobile, walking away from the base station, with the SNR falling as the distance from the base station increases. In this case, if the modulation technique used in the 802.11 protocol operating between the base station and the user does not change, the BER will become unacceptably high as the SNR decreases, and eventually no transmitted frames will be received correctly.

For this reason, some 802.11 implementations have a rate adaptation capability that adaptively selects the underlying physical-layer modulation technique to use based on current or recent channel characteristics. If a node sends two frames in a row without receiving an acknowledgment (an implicit indication of bit errors on

the channel), the transmission rate falls back to the next lower rate. If 10 frames in a row are acknowledged, or if a timer that tracks the time since the last fallback expires, the transmission rate increases to the next higher rate. This rate adaptation mechanism shares the same "probing" philosophy as TCP's congestion-control mechanism—when conditions are good (reflected by ACK receipts), the transmission rate is increased until something "bad" happens (the lack of ACK receipts); when something "bad" happens, the transmission rate is reduced. 802.11 rate adaptation and TCP congestion control are thus similar to the young child who is constantly pushing his/her parents for more and more (say candy for a young child, later curfew hours for the teenager) until the parents finally say "Enough!" and the child backs off (only to try again later after conditions have hopefully improved!). A number of other schemes have also been proposed to improve on this basic automatic rate-adjustment scheme [Kamerman 1997; Holland 2001; Lacage 2004].

Power Management

Power is a precious resource in mobile devices, and thus the 802.11 standard provides power-management capabilities that allow 802.11 nodes to minimize the amount of time that their sense, transmit, and receive functions and other circuitry need to be "on." 802.11 power management operates as follows. A node is able to explicitly alternate between sleep and wake states (not unlike a sleepy student in a classroom!). A node indicates to the access point that it will be going to sleep by setting the power-management bit in the header of an 802.11 frame to 1. A timer in the node is then set to wake up the node just before the AP is scheduled to send its beacon frame (recall that an AP typically sends a beacon frame every 100 msec). Since the AP knows from the set power-transmission bit that the node is going to sleep, it (the AP) knows that it should not send any frames to that node, and will buffer any frames destined for the sleeping host for later transmission.

A node will wake up just before the AP sends a beacon frame, and quickly enter the fully active state (unlike the sleepy student, this wakeup requires only 250 microseconds [Kamerman 1997]!). The beacon frames sent out by the AP contain a list of nodes whose frames have been buffered at the AP. If there are no buffered frames for the node, it can go back to sleep. Otherwise, the node can explicitly request that the buffered frames be sent by sending a polling message to the AP. With an inter-beacon time of 100 msec, a wakeup time of 250 microseconds, and a similarly small time to receive a beacon frame and check to ensure that there are no buffered frames, a node that has no frames to send or receive can be asleep 99% of the time, resulting in a significant energy savings.

7.3.6 Personal Area Networks: Bluetooth and Zigbee

As illustrated in Figure 7.2, the IEEE 802.11 WiFi standard is aimed at communication among devices separated by up to 100 meters (except when 802.11 is

used in a point-to-point configuration with a directional antenna). Two other wireless protocols in the IEEE 802 family are Bluetooth and Zigbee (defined in the IEEE 802.15.1 and IEEE 802.15.4 standards [IEEE 802.15 2012]).

Bluetooth

An IEEE 802.15.1 network operates over a short range, at low power, and at low cost. It is essentially a low-power, short-range, low-rate "cable replacement" technology for interconnecting a computer with its wireless keyboard, mouse or other peripheral device; cellular phones, speakers, headphones, and many other devices, whereas 802.11 is a higher-power, medium-range, higher-rate "access" technology. For this reason, 802.15.1 networks are sometimes referred to as wireless personal area networks (WPANs). The link and physical layers of 802.15.1 are based on the earlier **Bluetooth** specification for personal area networks [Held 2001, Bisdikian 2001]. 802.15.1 networks operate in the 2.4 GHz unlicensed radio band in a TDM manner, with time slots of 625 microseconds. During each time slot, a sender transmits on one of 79 channels, with the channel changing in a known but pseudo-random manner from slot to slot. This form of channel hopping, known as **frequency-hopping spread spectrum (FHSS)**, spreads transmissions in time over the frequency spectrum. 802.15.1 can provide data rates up to 4 Mbps.

802.15.1 networks are ad hoc networks: No network infrastructure (e.g., an access point) is needed to interconnect 802.15.1 devices. Thus, 802.15.1 devices must organize themselves. 802.15.1 devices are first organized into a **piconet** of up to eight active devices, as shown in Figure 7.16. One of these devices is designated as the master, with the remaining devices acting as slaves. The master node truly rules the piconet—its clock determines time in the piconet, it can transmit in each odd-numbered slot, and a

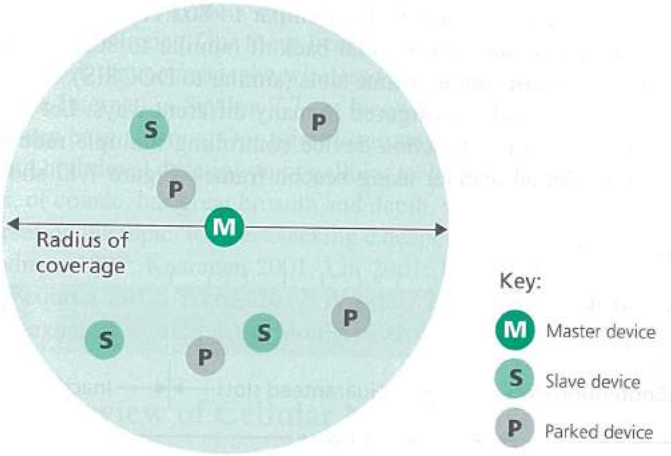


Figure 7.16 ♦ A Bluetooth piconet

slave can transmit only after the master has communicated with it in the previous slot and even then the slave can only transmit to the master. In addition to the slave devices, there can also be up to 255 parked devices in the network. These devices cannot communicate until their status has been changed from parked to active by the master node.

For more information about WPANs, the interested reader should consult the Bluetooth references [Held 2001, Bisdikian 2001] or the official IEEE 802.15 Web site [IEEE 802.15 2012].

Zigbee

A second personal area network standardized by the IEEE is the 802.15.4 standard [IEEE 802.15 2012] known as Zigbee. While Bluetooth networks provide a "cable replacement" data rate of over a Megabit per second, Zigbee is targeted at lower-powered, lower-data-rate, lower-duty-cycle applications than Bluetooth. While we may tend to think that "bigger and faster is better," not all network applications need high bandwidth and the consequent higher costs (both economic and power costs). For example, home temperature and light sensors, security devices, and wall-mounted switches are all very simple, low-power, low-duty-cycle, low-cost devices. Zigbee is thus well-suited for these devices. Zigbee defines channel rates of 20, 40, 100, and 250 Kbps, depending on the channel frequency.

Nodes in a Zigbee network come in two flavors. So-called "reduced-function devices" operate as slave devices under the control of a single "full-function device," much as Bluetooth slave devices. A full-function device can operate as a master device as in Bluetooth by controlling multiple slave devices, and multiple full-function devices can additionally be configured into a mesh network in which full-function devices route frames amongst themselves. Zigbee shares many protocol mechanisms that we've already encountered in other link-layer protocols: beacon frames and link-layer acknowledgments (similar to 802.11), carrier-sense random access protocols with binary exponential backoff (similar to 802.11 and Ethernet), and fixed, guaranteed allocation of time slots (similar to DOCSIS).

Zigbee networks can be configured in many different ways. Let's consider the simple case of a single full-function device controlling multiple reduced-function devices in a time-slotted manner using beacon frames. Figure 7.17 shows the case

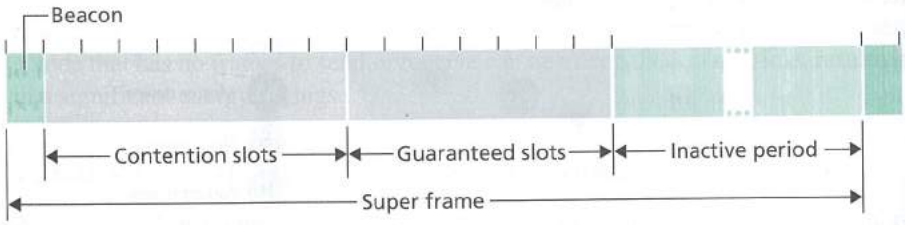


Figure 7.17 ♦ Zigbee 802.15.4 super-frame structure

where the Zigbee network divides time into recurring super frames, each of which begins with a beacon frame. Each beacon frame divides the super frame into an active period (during which devices may transmit) and an inactive period (during which all devices, including the controller, can sleep and thus conserve power). The active period consists of 16 time slots, some of which are used by devices in a CSMA/CA random access manner, and some of which are allocated by the controller to specific devices, thus providing guaranteed channel access for those devices. More details about Zigbee networks can be found at [Baronti 2007, IEEE 802.15.4 2012].

7.4 Cellular Internet Access

In the previous section we examined how an Internet host can access the Internet when inside a WiFi hotspot—that is, when it is within the vicinity of an 802.11 access point. But most WiFi hotspots have a small coverage area of between 10 and 100 meters in diameter. What do we do then when we have a desperate need for wireless Internet access and we cannot access a WiFi hotspot?

Given that cellular telephony is now ubiquitous in many areas throughout the world, a natural strategy is to extend cellular networks so that they support not only voice telephony but wireless Internet access as well. Ideally, this Internet access would be at a reasonably high speed and would provide for seamless mobility, allowing users to maintain their TCP sessions while traveling, for example, on a bus or a train. With sufficiently high upstream and downstream bit rates, the user could even maintain video-conferencing sessions while roaming about. This scenario is not that far-fetched. Data rates of several megabits per second are becoming available as broadband data services such as those we will cover here become more widely deployed.

In this section, we provide a brief overview of current and emerging cellular Internet access technologies. Our focus here will be on both the wireless first hop as well as the network that connects the wireless first hop into the larger telephone network and/or the Internet; in Section 7.7 we'll consider how calls are routed to a user moving between base stations. Our brief discussion will necessarily provide only a simplified and high-level description of cellular technologies. Modern cellular communications, of course, has great breadth and depth, with many universities offering several courses on the topic. Readers seeking a deeper understanding are encouraged to see [Goodman 1997; Kaaranen 2001; Lin 2001; Korhonen 2003; Schiller 2003; Palat 2009; Scourias 2012; Turner 2012; Akyildiz 2010], as well as the particularly excellent and exhaustive references [Mouly 1992; Sauter 2014].

7.4.1 An Overview of Cellular Network Architecture

In our description of cellular network architecture in this section, we'll adopt the terminology of the *Global System for Mobile Communications* (***GSM***) standards.

(For history buffs, the GSM acronym was originally derived from *Groupe Spécial Mobile*, until the more anglicized name was adopted, preserving the original acronym letters.) In the 1980s, Europeans recognized the need for a pan-European digital cellular telephony system that would replace the numerous incompatible analog cellular telephony systems, leading to the GSM standard [Mouly 1992]. Europeans deployed GSM technology with great success in the early 1990s, and since then GSM has grown to be the 800-pound gorilla of the cellular telephone world, with more than 80% of all cellular subscribers worldwide using GSM.

CASE HISTORY

4G CELLULAR MOBILE VERSUS WIRELESS LANS

Many cellular mobile phone operators are deploying 4G cellular mobile systems. In some countries (e.g., Korea and Japan), 4G LTE coverage is higher than 90%—nearly ubiquitous. In 2015, average download rates over deployed LTE systems range from 10Mbps in the US and India to close to 40 Mbps in New Zealand. These 4G systems are being deployed in licensed radio-frequency bands, with some operators paying considerable sums to governments for spectrum-use licenses. 4G systems allow users to access the Internet from remote outdoor locations while on the move, in a manner similar to today's cellular phone-only access. In many cases, a user may have simultaneous access to both wireless LANs and 4G. With the capacity of 4G systems being both more constrained and more expensive, many mobile devices default to the use of WiFi rather than 4G, when both are avilable. The question of whether wireless edge network access will be primarily over wireless LANs or cellular systems remains an open question:

- The emerging wireless LAN infrastructure may become nearly ubiquitous. IEEE 802.11 wireless LANs, operating at 54 Mbps and higher, are enjoying widespread deployment. Essentially all laptops, tablets and smartphones are factory-equipped with 802.11 LAN capabilities. Furthermore, emerging Internet appliances—such as wireless cameras and picture frames—also have low-powered wireless LAN capabilities.
- Wireless LAN base stations can also handle mobile phone appliances. Many phones are already capable of connecting to the cellular phone network or to an IP network either natively or using a Skype-like Voice-over-IP service, thus bypassing the operator's cellular voice and 4G data services.

Of course, many other experts believe that 4G not only will be a major success, but will also dramatically revolutionize the way we work and live. Most likely, both WiFi and 4G will both become prevalent wireless technologies, with roaming wireless devices automatically selecting the access technology that provides the best service at their current physical location.

When people talk about cellular technology, they often classify the technology as belonging to one of several "generations." The earliest generations were designed primarily for voice traffic. First generation (1G) systems were analog FDMA systems designed exclusively for voice-only communication. These 1G systems are almost extinct now, having been replaced by digital 2G systems. The original 2G systems were also designed for voice, but later extended (2.5G) to support data (i.e., Internet) as well as voice service. 3G systems also support voice and data, but with an emphasis on data capabilities and higher-speed radio access links. The 4G systems being deployed today are based on LTE technology, feature an all-IP core network, and provide integrated voice and data at multi-Megabit speeds.

Cellular Network Architecture, 2G: Voice Connections to the Telephone Network

The term *cellular* refers to the fact that the region covered by a cellular network is partitioned into a number of geographic coverage areas, known as **cells**, shown as hexagons on the left side of Figure 7.18. As with the 802.11WiFi standard we studied in Section 7.3.1, GSM has its own particular nomenclature. Each cell

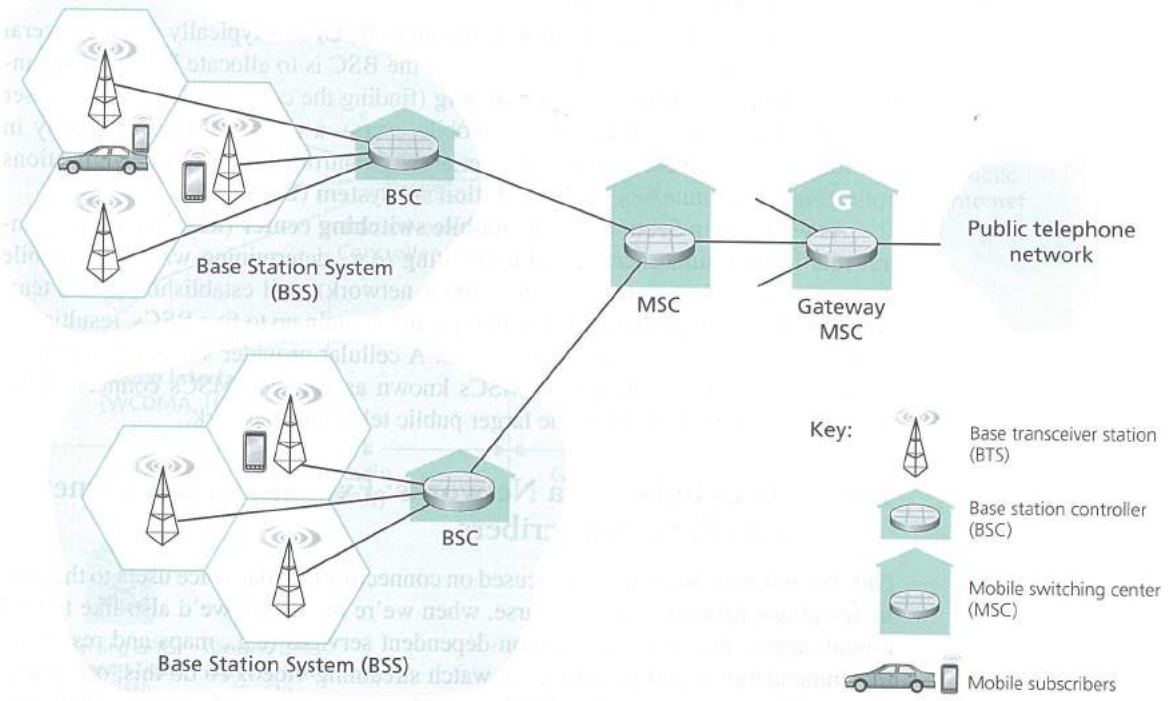


Figure 7.18 ♦ Components of the GSM 2G cellular network architecture

contains a **base transceiver station (BTS)** that transmits signals to and receives signals from the mobile stations in its cell. The coverage area of a cell depends on many factors, including the transmitting power of the BTS, the transmitting power of the user devices, obstructing buildings in the cell, and the height of base station antennas. Although Figure 7.18 shows each cell containing one base transceiver station residing in the middle of the cell, many systems today place the BTS at corners where three cells intersect, so that a single BTS with directional antennas can service three cells.

The GSM standard for 2G cellular systems uses combined FDM/TDM (radio) for the air interface. Recall from Chapter 1 that, with pure FDM, the channel is partitioned into a number of frequency bands with each band devoted to a call. Also recall from Chapter 1 that, with pure TDM, time is partitioned into frames with each frame further partitioned into slots and each call being assigned the use of a particular slot in the revolving frame. In combined FDM/TDM systems, the channel is partitioned into a number of frequency sub-bands; within each sub-band, time is partitioned into frames and slots. Thus, for a combined FDM/TDM system, if the channel is partitioned into F sub-bands and time is partitioned into T slots, then the channel will be able to support $F.T$ simultaneous calls. Recall that we saw in Section 6.3.4 that cable access networks also use a combined FDM/TDM approach. GSM systems consist of 200-kHz frequency bands with each band supporting eight TDM calls. GSM encodes speech at 13 kbps and 12.2 kbps.

A GSM network's **base station controller (BSC)** will typically service several tens of base transceiver stations. The role of the BSC is to allocate BTS radio channels to mobile subscribers, perform **paging** (finding the cell in which a mobile user is resident), and perform handoff of mobile users—a topic we'll cover shortly in Section 7.7.2. The base station controller and its controlled base transceiver stations collectively constitute a GSM **base station subsystem (BSS)**.

As we'll see in Section 7.7, the **mobile switching center (MSC)** plays the central role in user authorization and accounting (e.g., determining whether a mobile device is allowed to connect to the cellular network), call establishment and teardown, and handoff. A single MSC will typically contain up to five BSCs, resulting in approximately 200K subscribers per MSC. A cellular provider's network will have a number of MSCs, with special MSCs known as gateway MSCs connecting the provider's cellular network to the larger public telephone network.

7.4.2 3G Cellular Data Networks: Extending the Internet to Cellular Subscribers

Our discussion in Section 7.4.1 focused on connecting cellular voice users to the public telephone network. But, of course, when we're on the go, we'd also like to read e-mail, access the Web, get location-dependent services (e.g., maps and restaurant recommendations) and perhaps even watch streaming video. To do this, our smartphone will need to run a full TCP/IP protocol stack (including the physical link, network, transport, and application layers) and connect into the Internet via the cellular

data network. The topic of cellular data networks is a rather bewildering collection of competing and ever-evolving standards as one generation (and half-generation) succeeds the former and introduces new technologies and services with new acronyms. To make matters worse, there's no single official body that sets requirements for 2.5G, 3G, 3.5G, or 4G technologies, making it hard to sort out the differences among competing standards. In our discussion below, we'll focus on the UMTS (Universal Mobile Telecommunications Service) 3G and 4G standards developed by the 3rd Generation Partnership project (3GPP) [3GPP 2016].

Let's first take a top-down look at 3G cellular data network architecture shown in Figure 7.19.

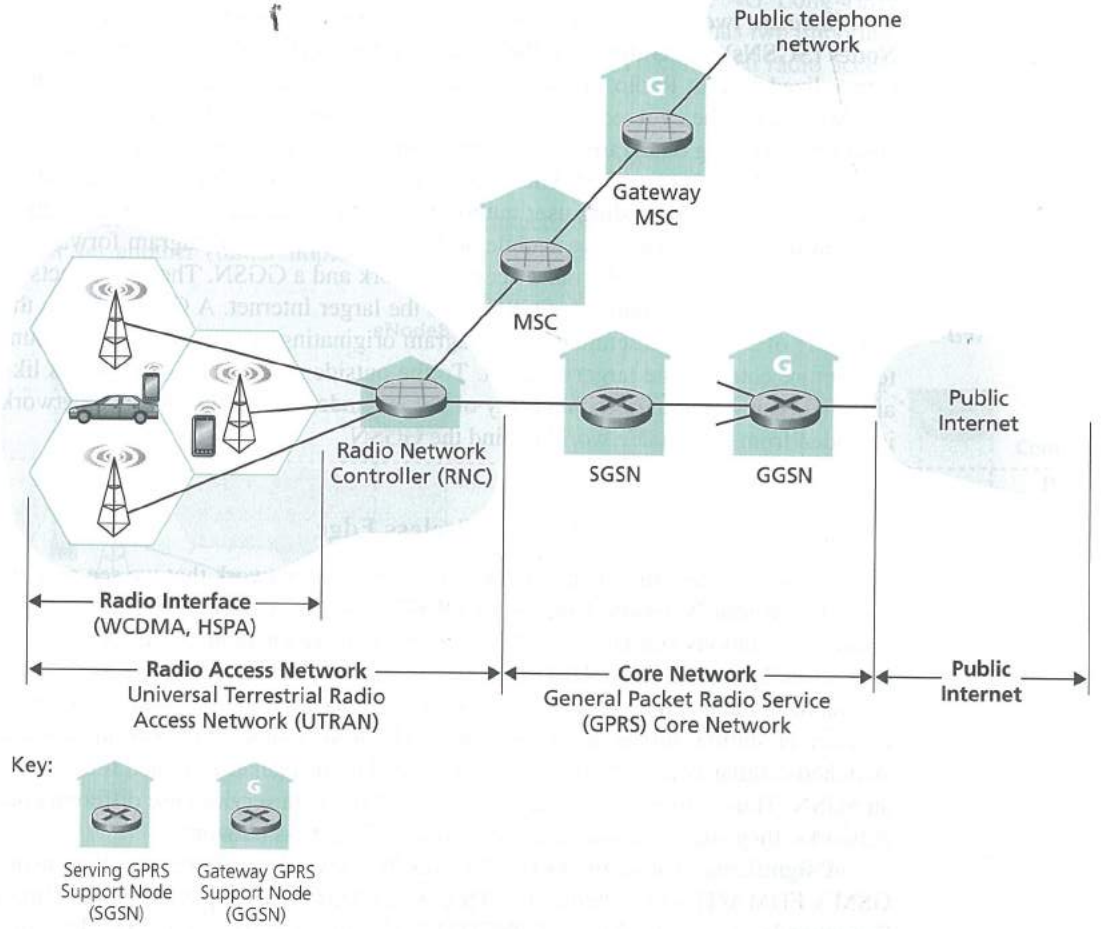


Figure 7.19 ♦ 3G system architecture

3G Core Network

The 3G core cellular data network connects radio access networks to the public Internet. The core network interoperates with components of the existing cellular voice network (in particular, the MSC) that we previously encountered in Figure 7.18. Given the considerable amount of existing infrastructure (and profitable services!) in the existing cellular voice network, the approach taken by the designers of 3G data services is clear: *leave the existing core GSM cellular voice network untouched, adding additional cellular data functionality in parallel to the existing cellular voice network*. The alternative—integrating new data services directly into the core of the existing cellular voice network—would have raised the same challenges encountered in Section 4.3, where we discussed integrating new (IPv6) and legacy (IPv4) technologies in the Internet.

There are two types of nodes in the 3G core network: **Serving GPRS Support Nodes (SGSNs)** and **Gateway GPRS Support Nodes (GGSNs)**. (GPRS stands for Generalized Packet Radio Service, an early cellular data service in 2G networks; here we discuss the evolved version of GPRS in 3G networks). An SGSN is responsible for delivering datagrams to/from the mobile nodes in the radio access network to which the SGSN is attached. The SGSN interacts with the cellular voice network's MSC for that area, providing user authorization and handoff, maintaining location (cell) information about active mobile nodes, and performing datagram forwarding between mobile nodes in the radio access network and a GGSN. The GGSN acts as a gateway, connecting multiple SGSNs into the larger Internet. A GGSN is thus the last piece of 3G infrastructure that a datagram originating at a mobile node encounters before entering the larger Internet. To the outside world, the GGSN looks like any other gateway router; the mobility of the 3G nodes within the GGSN's network is hidden from the outside world behind the GGSN.

3G Radio Access Network: The Wireless Edge

The 3G **radio access network** is the wireless first-hop network that we see as a 3G user. The **Radio Network Controller (RNC)** typically controls several cell base transceiver stations similar to the base stations that we encountered in 2G systems (but officially known in 3G UMTS parlance as a "Node Bs"—a rather non-descriptive name!). Each cell's wireless link operates between the mobile nodes and a base transceiver station, just as in 2G networks. The RNC connects to both the circuit-switched cellular voice network via an MSC, and to the packet-switched Internet via an SGSN. Thus, while 3G cellular voice and cellular data services use different core networks, they share a common first/last-hop radio access network.

A significant change in 3G UMTS over 2G networks is that rather than using GSM's FDMA/TDMA scheme, UMTS uses a CDMA technique known as Direct Sequence Wideband CDMA (DS-WCDMA) [Dahlman 1998] within TDMA slots; TDMA slots, in turn, are available on multiple frequencies—an interesting use of

all three dedicated channel-sharing approaches that we earlier identified in Chapter 6 and similar to the approach taken in wired cable access networks (see Section 6.3.4). This change requires a new 3G cellular wireless-access network operating in parallel with the 2G BSS radio network shown in Figure 7.19. The data service associated with the WCDMA specification is known as HSPA (High Speed Packet Access) and promises downlink data rates of up to 14 Mbps. Details regarding 3G networks can be found at the 3rd Generation Partnership Project (3GPP) Web site [3GPP 2016].

7.4.3 On to 4G: LTE

Fourth generation (4G) cellular systems are becoming widely deployed. In 2015, more than 50 countries had 4G coverage exceeding 50%. The 4G Long-Term Evolution (LTE) standard [Sauter 2014] put forward by the 3GPP has two important innovations over 3G systems an all-IP core network and an enhanced radio access network, as discussed below.

4G System Architecture: An All-IP Core Network

Figure 7.20 shows the overall 4G network architecture, which (unfortunately) introduces yet another (rather impenetrable) new vocabulary and set of acronyms for

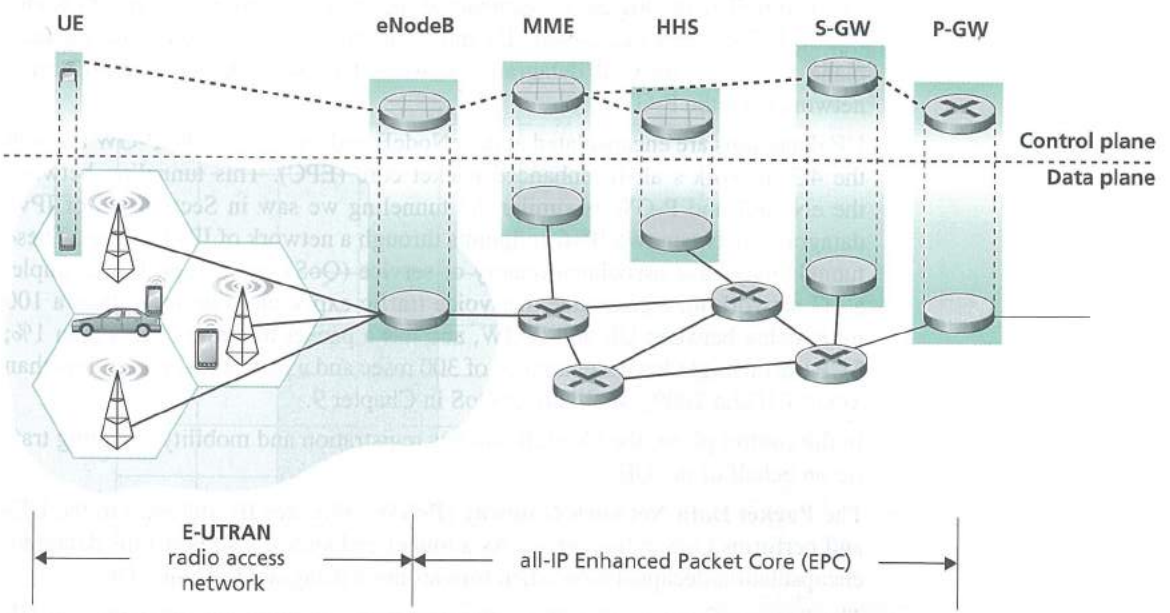


Figure 7.20 ♦ 4G network architecture

network components. But let's not get lost in these acronyms! There are two important high-level observations about the 4G architecture:

- *A unified, all-IP network architecture.* Unlike the 3G network shown in Figure 7.19, which has separate network components and paths for voice and data traffic, the 4G architecture shown in Figure 7.20 is "all-IP"—both voice and data are carried in IP datagrams to/from the wireless device (the User Equipment, UE in 4G parlance) to the gateway to the packet gateway (P-GW) that connects the 4G edge network to the rest of the network. With 4G, the last vestiges of cellular networks' roots in the telephony have disappeared, giving way to universal IP service!
- *A clear separation of the 4G data plane and 4G control plane.* Mirroring our distinction between the data and control planes for IP's network layer in Chapters 4 and 5 respectively, the 4G network architecture also clearly separates the data and control planes. We'll discuss their functionality below.
- *A clear separation between the radio access network, and the all-IP-core network.* IP datagrams carrying user data are forwarded between the user (UE) and the gateway (P-GW in Figure 7.20) over a 4G-internal IP network to the external Internet. Control packets are exchanged over this same internal network among the 4G's control services components, whose roles are described below.

The principal components of the 4G architecture are as follows.

- The **eNodeB** is the logical descendant of the 2G base station and the 3G Radio Network Controller (a.k.a Node B) and again plays a central role here. Its data-plane role is to forward datagrams between UE (over the LTE radio access network) and the P-GW.

  UE datagrams are encapsulated at the eNodeB and tunneled to the P-GW through the 4G network's all-IP enhanced packet core (EPC). This tunneling between the eNodeB and P-GW is similar the tunneling we saw in Section 4.3 of IPv6 datagrams between two IPv6 endpoints through a network of IPv4 routers. These tunnels may have associated quality of service (QoS) guarantees. For example, a 4G network may guarantee that voice traffic experiences no more than a 100 msec delay between UE and P-GW, and has a packet loss rate of less than 1%; TCP traffic might have a guarantee of 300 msec and a packet loss rate of less than .0001% [Palat 2009]. We'll cover QoS in Chapter 9.

  In the control plane, the eNodeB handles registration and mobility signaling traffic on behalf of the UE.
- The **Packet Data Network Gateway (P-GW)** allocates IP addresses to the UEs and performs QoS enforcement. As a tunnel endpoint it also performs datagram encapsulation/decapsulation when forwarding a datagram to/from a UE.
- The Serving Gateway (S-GW) is the data-plane mobility anchor point—all UE traffic will pass through the S-GW. The S-GW also performs charging/billing functions and lawful traffic interception.

- The **Mobility Management Entity (MME)** performs connection and mobility management on behalf of the UEs resident in the cell it controls. It receives UE subscription information from the HHS. We cover mobility in cellular networks in detail in Section 7.7.
- **The Home Subscriber Server (HSS)** contains UE information including roaming access capabilities, quality of service profiles, and authentication information. As we'll see in Section 7.7, the HSS obtains this information from the UE's home cellular provider.

Very readable introductions to 4G network architecture and its EPC are [Motorola 2007; Palat 2009; Sauter 2014].

LTE Radio Access Network

LTE uses a combination of frequency division multiplexing and time division multiplexing on the downstream channel, known as orthogonal frequency division multiplexing (OFDM) [Rohde 2008; Ericsson 2011]. (The term "orthogonal" comes from the fact the signals being sent on different frequency channels are created so that they interfere very little with each other, even when channel frequencies are tightly spaced). In LTE, each active mobile node is allocated one or more 0.5 ms time slots in one or more of the channel frequencies. Figure 7.21 shows an allocation of eight time slots over four frequencies. By being allocated increasingly more time slots (whether on the same frequency or on different frequencies), a mobile node is able to achieve increasingly higher transmission rates. Slot (re)allocation among mobile

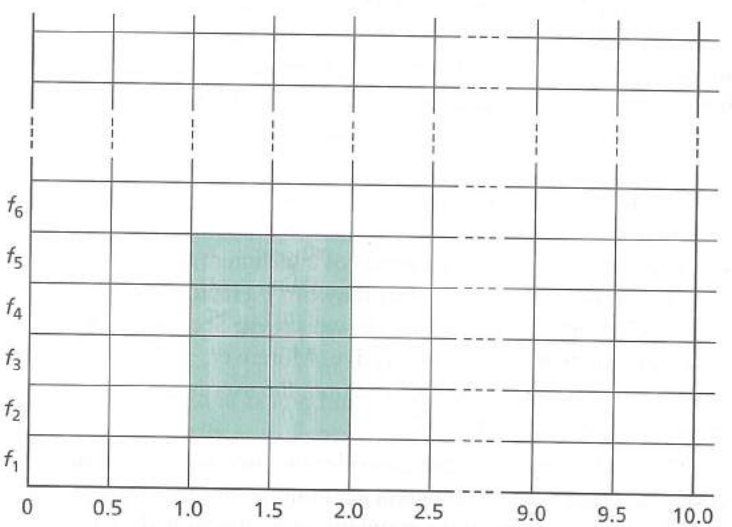


Figure 7.21 ♦ Twenty 0.5 ms slots organized into 10 ms frames at each frequency. An eight-slot allocation is shown shaded.

nodes can be performed as often as once every millisecond. Different modulation schemes can also be used to change the transmission rate; see our earlier discussion of Figure 7.3 and dynamic selection of modulation schemes in WiFi networks.

The particular allocation of time slots to mobile nodes is not mandated by the LTE standard. Instead, the decision of which mobile nodes will be allowed to transmit in a given time slot on a given frequency is determined by the scheduling algorithms provided by the LTE equipment vendor and/or the network operator. With opportunistic scheduling [Bender 2000; Kolding 2003; Kulkarni 2005], matching the physical-layer protocol to the channel conditions between the sender and receiver and choosing the receivers to which packets will be sent based on channel conditions allow the radio network controller to make best use of the wireless medium. In addition, user priorities and contracted levels of service (e.g., silver, gold, or platinum) can be used in scheduling downstream packet transmissions. In addition to the LTE capabilities described above, LTE-Advanced allows for downstream bandwidths of hundreds of Mbps by allocating aggregated channels to a mobile node [Akyildiz 2010].

An additional 4G wireless technology—WiMAX (World Interoperability for Microwave Access)—is a family of IEEE 802.16 standards that differ significantly from LTE. WiMAX has not yet been able to enjoy the widespread deployment of LTE. A detailed discussion of WiMAX can be found on this book's Web site.

7.5 Mobility Management: Principles

Having covered the *wireless* nature of the communication links in a wireless network, it's now time to turn our attention to the *mobility* that these wireless links enable. In the broadest sense, a mobile node is one that changes its point of attachment into the network over time. Because the term *mobility* has taken on many meanings in both the computer and telephony worlds, it will serve us well first to consider several dimensions of mobility in some detail.

- *From the network layer's standpoint, how mobile is a user?* A physically mobile user will present a very different set of challenges to the network layer, depending on how he or she moves between points of attachment to the network. At one end of the spectrum in Figure 7.22, a user may carry a laptop with a wireless network interface card around in a building. As we saw in Section 7.3.4, this user is *not* mobile from a network-layer perspective. Moreover, if the user associates with the same access point regardless of location, the user is not even mobile from the perspective of the link layer.

  At the other end of the spectrum, consider the user zooming along the autobahn in a BMW or Tesla at 150 kilometers per hour, passing through multiple wireless access networks and wanting to maintain an uninterrupted TCP connection to a remote application throughout the trip. This user is *definitely* mobile! In between

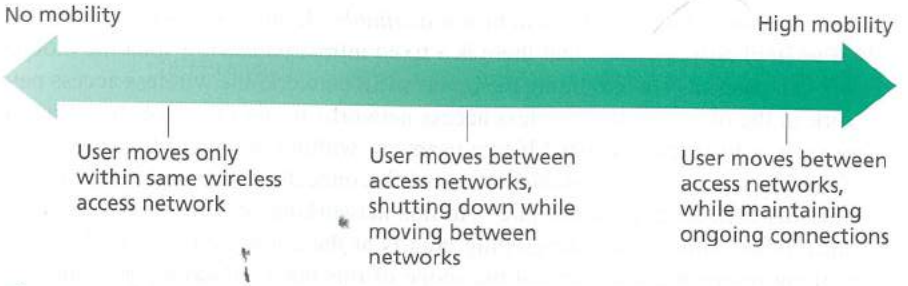


Figure 7.22 ♦ Various degrees of mobility, from the network layer's point of view

these extremes is a user who takes a laptop from one location (e.g., office or dormitory) into another (e.g., coffeeshop, classroom) and wants to connect into the-network in the new location. This user is also mobile (although less so than the BMW driver!) but does not need to maintain an ongoing connection while moving between points of attachment to the network. Figure 7.22 illustrates this spectrum of user mobility from the network layer's perspective.

- *How important is it for the mobile node's address to always remain the same?* With mobile telephony, your phone number—essentially the network-layer address of your phone—remains the same as you travel from one provider's mobile phone network to another. Must a laptop similarly maintain the same IP address while moving between IP networks?

  The answer to this question will depend strongly on the applications being run. For the BMW or Tesla driver who wants to maintain an uninterrupted TCP connection to a remote application while zipping along the autobahn, it would be convenient to maintain the same IP address. Recall from Chapter 3 that an Internet application needs to know the IP address and port number of the remote entity with which it is communicating. If a mobile entity is able to maintain its IP address as it moves, mobility becomes invisible from the application standpoint. There is great value to this transparency—an application need not be concerned with a potentially changing IP address, and the same application code serves mobile and nonmobile connections alike. We'll see in the following section that mobile IP provides this transparency, allowing a mobile node to maintain its permanent IP address while moving among networks.

  On the other hand, a less glamorous mobile user might simply want to turn off an office laptop, bring that laptop home, power up, and work from home. If the laptop functions primarily as a client in client-server applications (e.g., send/read e-mail, browse the Web, Telnet to a remote host) from home, the particular IP address used by the laptop is not that important. In particular, one could get by fine with an address that is temporarily allocated to the laptop by the ISP serving the home. We saw in Section 4.3 that DHCP already provides this functionality.

- *What supporting wired infrastructure is available?* In all of our scenarios above, we've implicitly assumed that there is a fixed infrastructure to which the mobile user can connect—for example, the home's ISP network, the wireless access network in the office, or the wireless access networks lining the autobahn. What if no such infrastructure exists? If two users are within communication proximity of each other, can they establish a network connection in the absence of any other network-layer infrastructure? Ad hoc networking provides precisely these capabilities. This rapidly developing area is at the cutting edge of mobile networking research and is beyond the scope of this book. [Perkins 2000] and the IETF Mobile Ad Hoc Network (manet) working group Web pages [manet 2016] provide thorough treatments of the subject.

In order to illustrate the issues involved in allowing a mobile user to maintain ongoing connections while moving between networks, let's consider a human analogy. A twenty-something adult moving out of the family home becomes mobile, living in a series of dormitories and/or apartments, and often changing addresses. If an old friend wants to get in touch, how can that friend find the address of her mobile friend? One common way is to contact the family, since a mobile adult will often register his or her current address with the family (if for no other reason than so that the parents can send money to help pay the rent!). The family home, with its permanent address, becomes that one place that others can go as a first step in communicating with the mobile adult. Later communication from the friend may be either indirect (for example, with mail being sent first to the parents' home and then forwarded to the mobile adult) or direct (for example, with the friend using the address obtained from the parents to send mail directly to her mobile friend).

In a network setting, the permanent home of a mobile node (such as a laptop or smartphone) is known as the **home network**, and the entity within the home network that performs the mobility management functions discussed below on behalf of the mobile node is known as the **home agent**. The network in which the mobile node is currently residing is known as the **foreign** (or **visited**) **network**, and the entity within the foreign network that helps the mobile node with the mobility management functions discussed below is known as a **foreign agent**. For mobile professionals, their home network might likely be their company network, while the visited network might be the network of a colleague they are visiting. A **correspondent** is the entity wishing to communicate with the mobile node. Figure 7.23 illustrates these concepts, as well as addressing concepts considered below. In Figure 7.23, note that agents are shown as being collocated with routers (e.g., as processes running on routers), but alternatively they could be executing on other hosts or servers in the network.

7.5.1 Addressing

We noted above that in order for user mobility to be transparent to network applications, it is desirable for a mobile node to keep its address as it moves from one network

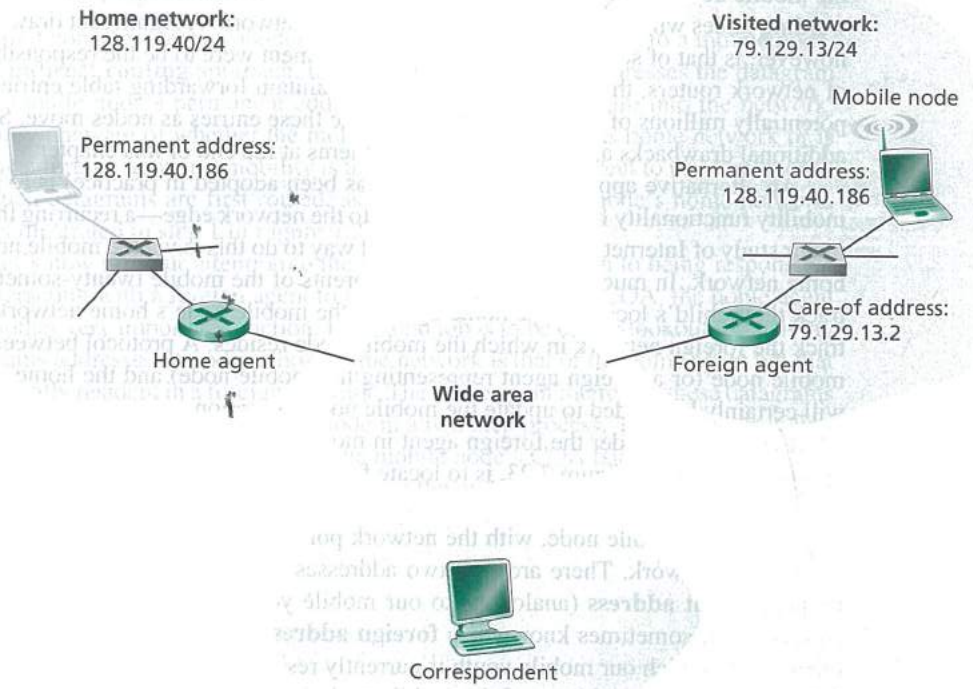


Figure 7.23 ♦ Initial elements of a mobile network architecture

to another. When a mobile node is resident in a foreign network, all traffic addressed to the node's permanent address now needs to be routed to the foreign network. How can this be done? One option is for the foreign network to advertise to all other networks that the mobile node is resident in its network. This could be via the usual exchange of intradomain and interdomain routing information and would require few changes to the existing routing infrastructure. The foreign network could simply advertise to its neighbors that it has a highly specific route to the mobile node's permanent address (that is, essentially inform other networks that it has the correct path for routing datagrams to the mobile node's permanent address; see Section 4.3). These neighbors would then propagate this routing information throughout the network as part of the normal procedure of updating routing information and forwarding tables. When the mobile node leaves one foreign network and joins another, the new foreign network would advertise a new, highly specific route to the mobile node, and the old foreign network would withdraw its routing information regarding the mobile node.

This solves two problems at once, and it does so without making significant changes to the network-layer infrastructure. Other networks know the location of

the mobile node, and it is easy to route datagrams to the mobile node, since the forwarding tables will direct datagrams to the foreign network. A significant drawback, however, is that of scalability. If mobility management were to be the responsibility of network routers, the routers would have to maintain forwarding table entries for potentially millions of mobile nodes, and update these entries as nodes move. Some additional drawbacks are explored in the problems at the end of this chapter.

An alternative approach (and one that has been adopted in practice) is to push mobility functionality from the network core to the network edge—a recurring theme in our study of Internet architecture. A natural way to do this is via the mobile node's home network. In much the same way that parents of the mobile twenty-something track their child's location, the home agent in the mobile node's home network can track the foreign network in which the mobile node resides. A protocol between the mobile node (or a foreign agent representing the mobile node) and the home agent will certainly be needed to update the mobile node's location.

Let's now consider the foreign agent in more detail. The conceptually simplest approach, shown in Figure 7.23, is to locate foreign agents at the edge routers in the foreign network. One role of the foreign agent is to create a so-called **care-of address (COA)** for the mobile node, with the network portion of the COA matching that of the foreign network. There are thus two addresses associated with a mobile node, its **permanent address** (analogous to our mobile youth's family's home address) and its COA, sometimes known as a **foreign address** (analogous to the address of the house in which our mobile youth is currently residing). In the example in Figure 7.23, the permanent address of the mobile node is 128.119.40.186. When visiting network 79.129.13/24, the mobile node has a COA of 79.129.13.2. A second role of the foreign agent is to inform the home agent that the mobile node is resident in its (the foreign agent's) network and has the given COA. We'll see shortly that the COA will be used to "reroute" datagrams to the mobile node via its foreign agent.

Although we have separated the functionality of the mobile node and the foreign agent, it is worth noting that the mobile node can also assume the responsibilities of the foreign agent. For example, the mobile node could obtain a COA in the foreign network (for example, using a protocol such as DHCP) and itself inform the home agent of its COA.

7.5.2 Routing to a Mobile Node

We have now seen how a mobile node obtains a COA and how the home agent can be informed of that address. But having the home agent know the COA solves only part of the problem. How should datagrams be addressed and forwarded to the mobile node? Since only the home agent (and not network-wide routers) knows the location of the mobile node, it will no longer suffice to simply address a datagram to the mobile node's permanent address and send it into the network-layer infrastructure. Something more must be done. Two approaches can be identified, which we will refer to as indirect and direct routing.

Indirect Routing to a Mobile Node

Let's first consider a correspondent that wants to send a datagram to a mobile node. In the **indirect routing** approach, the correspondent simply addresses the datagram to the mobile node's permanent address and sends the datagram into the network, blissfully unaware of whether the mobile node is resident in its home network or is visiting a foreign network; mobility is thus completely transparent to the correspondent. Such datagrams are first routed, as usual, to the mobile node's home network. This is illustrated in step 1 in Figure 7.24.

Let's now turn our attention to the home agent. In addition to being responsible for interacting with a foreign agent to track the mobile node's COA, the home agent has another very important function. Its second job is to be on the lookout for arriving datagrams addressed to nodes whose home network is that of the home agent but that are currently resident in a foreign network. The home agent intercepts these datagrams and then forwards them to a mobile node in a two-step process. The datagram is first forwarded to the foreign agent, using the mobile node's COA (step 2 in Figure 7.24), and then forwarded from the foreign agent to the mobile node (step 3 in Figure 7.24).

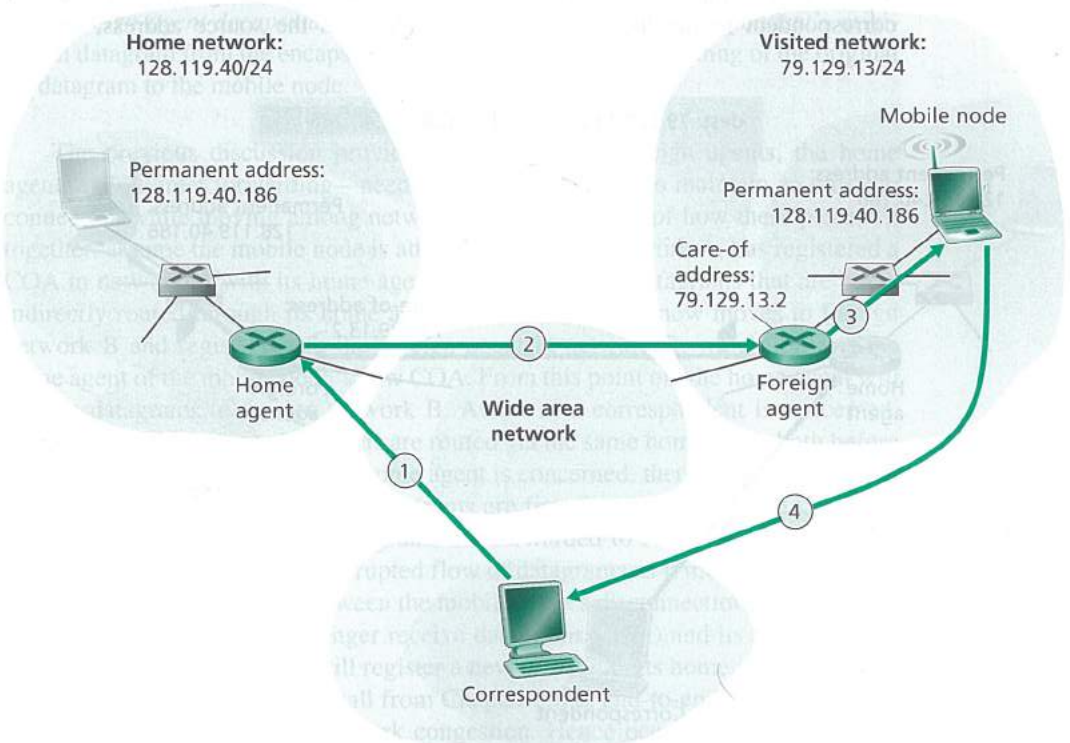


Figure 7.24 ♦ Indirect routing to a mobile node

It is instructive to consider this rerouting in more detail. The home agent will need to address the datagram using the mobile node's COA, so that the network layer will route the datagram to the foreign network. On the other hand, it is desirable to leave the correspondent's datagram intact, since the application receiving the datagram should be unaware that the datagram was forwarded via the home agent. Both goals can be satisfied by having the home agent **encapsulate** the correspondent's original complete datagram within a new (larger) datagram. This larger datagram is addressed and delivered to the mobile node's COA. The foreign agent, who "owns" the COA, will receive and decapsulate the datagram—that is, remove the correspondent's original datagram from within the larger encapsulating datagram and forward (step 3 in Figure 7.24) the original datagram to the mobile node. Figure 7.25 shows a correspondent's original datagram being sent to the home network, an encapsulated datagram being sent to the foreign agent, and the original datagram being delivered to the mobile node. The sharp reader will note that the encapsulation/decapsulation described here is identical to the notion of tunneling, discussed in Section 4.3 in the context of IP multicast and IPv6.

Let's next consider how a mobile node sends datagrams to a correspondent. This is quite simple, as the mobile node can address its datagram *directly* to the correspondent (using its own permanent address as the source address, and the

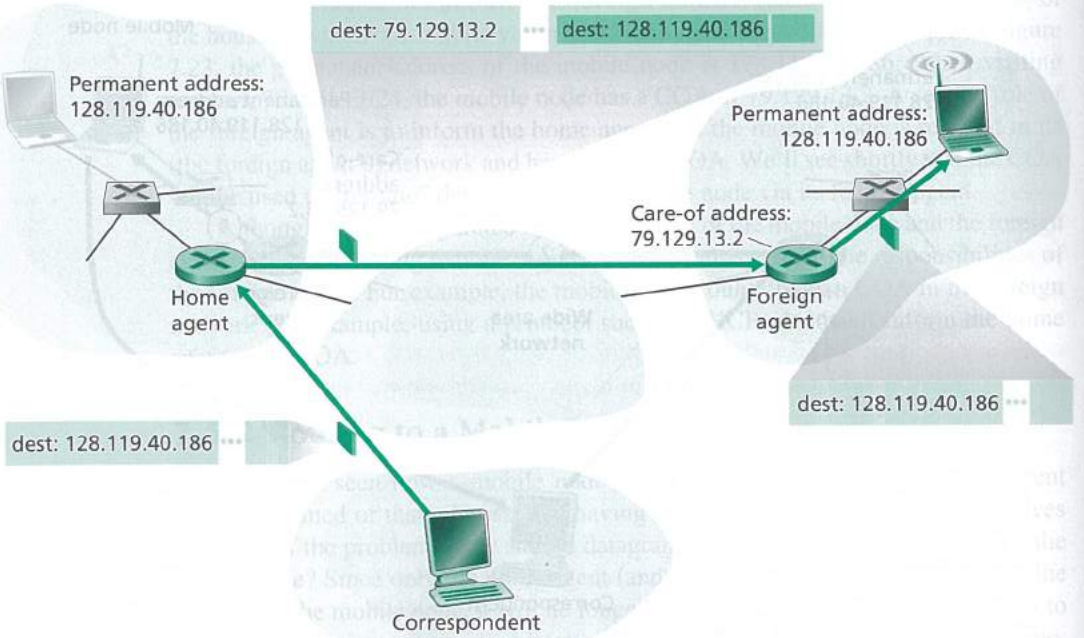


Figure 7.25 ♦ Encapsulation and decapsulation

correspondent's address as the destination address). Since the mobile node knows the correspondent's address, there is no need to route the datagram back through the home agent. This is shown as step 4 in Figure 7.24.

Let's summarize our discussion of indirect routing by listing the new network-layer functionality required to support mobility.

- *A mobile-node–to–foreign-agent protocol.* The mobile node will register with the foreign agent when attaching to the foreign network. Similarly, a mobile node will deregister with the foreign agent when it leaves the foreign network.
- *A foreign-agent–to–home-agent registration protocol.* The foreign agent will register the mobile node's COA with the home agent. A foreign agent need not explicitly deregister a COA when a mobile node leaves its network, because the subsequent registration of a new COA, when the mobile node moves to a new network, will take care of this.
- *A home-agent datagram encapsulation protocol.* Encapsulation and forwarding of the correspondent's original datagram within a datagram addressed to the COA.
- *A foreign-agent decapsulation protocol.* Extraction of the correspondent's original datagram from the encapsulating datagram, and the forwarding of the original datagram to the mobile node.

The previous discussion provides all the pieces—foreign agents, the home agent, and indirect forwarding—needed for a mobile node to maintain an ongoing connection while moving among networks. As an example of how these pieces fit together, assume the mobile node is attached to foreign network A, has registered a COA in network A with its home agent, and is receiving datagrams that are being indirectly routed through its home agent. The mobile node now moves to foreign network B and registers with the foreign agent in network B, which informs the home agent of the mobile node's new COA. From this point on, the home agent will reroute datagrams to foreign network B. As far as a correspondent is concerned, mobility is transparent—datagrams are routed via the same home agent both before and after the move. As far as the home agent is concerned, there is no disruption in the flow of datagrams—arriving datagrams are first forwarded to foreign network A; after the change in COA, datagrams are forwarded to foreign network B. But will the mobile node see an interrupted flow of datagrams as it moves between networks? As long as the time between the mobile node's disconnection from network A (at which point it can no longer receive datagrams via A) and its attachment to network B (at which point it will register a new COA with its home agent) is small, few datagrams will be lost. Recall from Chapter 3 that end-to-end connections can suffer datagram loss due to network congestion. Hence occasional datagram loss within a connection when a node moves between networks is by no means a catastrophic problem. If loss-free communication is required, upper-layer mechanisms

will recover from datagram loss, whether such loss results from network congestion or from user mobility.

An indirect routing approach is used in the mobile IP standard [RFC 5944], as discussed in Section 7.6.

Direct Routing to a Mobile Node

The indirect routing approach illustrated in Figure 7.24 suffers from an inefficiency known as the **triangle routing problem**—datagrams addressed to the mobile node must be routed first to the home agent and then to the foreign network, even when a much more efficient route exists between the correspondent and the mobile node. In the worst case, imagine a mobile user who is visiting the foreign network of a colleague. The two are sitting side by side and exchanging data over the network. Datagrams from the correspondent (in this case the colleague of the visitor) are routed to the mobile user's home agent and then back again to the foreign network!

Direct routing overcomes the inefficiency of triangle routing, but does so at the cost of additional complexity. In the direct routing approach, a **correspondent agent** in the correspondent's network first learns the COA of the mobile node. This can be done by having the correspondent agent query the home agent, assuming that (as in the case of indirect routing) the mobile node has an up-to-date value for its COA registered with its home agent. It is also possible for the correspondent itself to perform the function of the correspondent agent, just as a mobile node could perform the function of the foreign agent. This is shown as steps 1 and 2 in Figure 7.26. The correspondent agent then tunnels datagrams directly to the mobile node's COA, in a manner analogous to the tunneling performed by the home agent, steps 3 and 4 in Figure 7.26.

While direct routing overcomes the triangle routing problem, it introduces two important additional challenges:

- A **mobile-user location protocol** is needed for the correspondent agent to query the home agent to obtain the mobile node's COA (steps 1 and 2 in Figure 7.26).
- When the mobile node moves from one foreign network to another, how will data now be forwarded to the new foreign network? In the case of indirect routing, this problem was easily solved by updating the COA maintained by the home agent. However, with direct routing, the home agent is queried for the COA by the correspondent agent only once, at the beginning of the session. Thus, updating the COA at the home agent, while necessary, will not be enough to solve the problem of routing data to the mobile node's new foreign network.

One solution would be to create a new protocol to notify the correspondent of the changing COA. An alternate solution, and one that we'll see adopted in practice

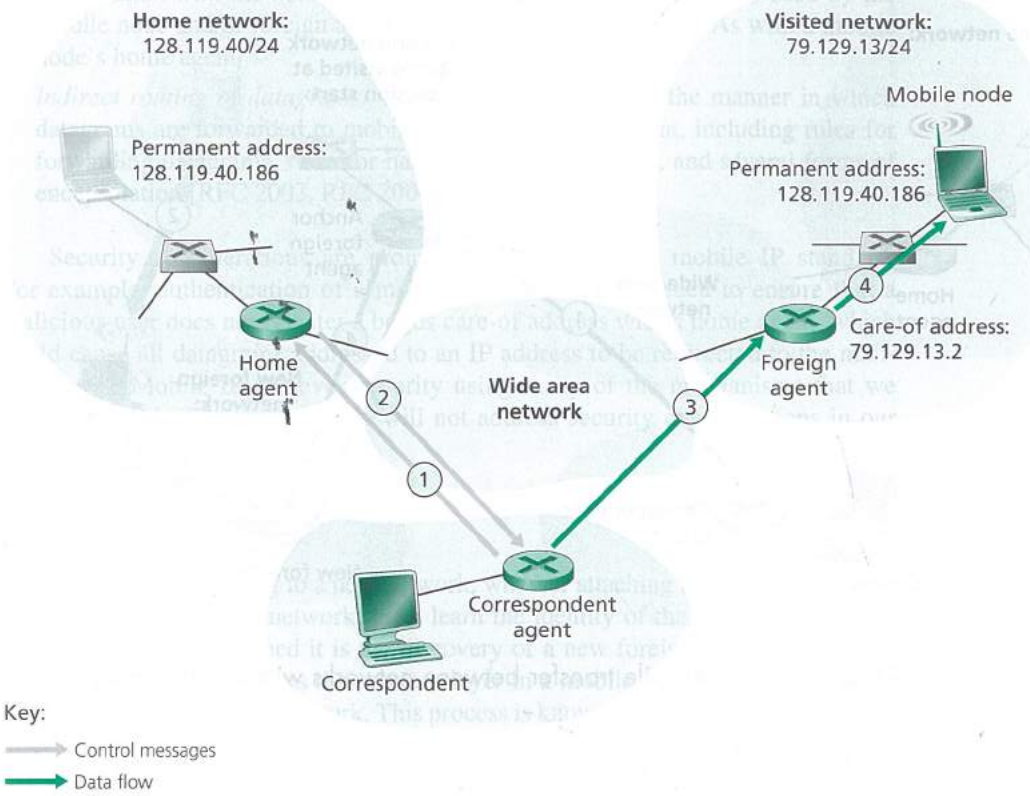


Figure 7.26 ♦ Direct routing to a mobile user

in GSM networks, works as follows. Suppose data is currently being forwarded to the mobile node in the foreign network where the mobile node was located when the session first started (step 1 in Figure 7.27). We'll identify the foreign agent in that foreign network where the mobile node was first found as the **anchor foreign agent**. When the mobile node moves to a new foreign network (step 2 in Figure 7.27), the mobile node registers with the new foreign agent (step 3), and the new foreign agent provides the anchor foreign agent with the mobile node's new COA (step 4). When the anchor foreign agent receives an encapsulated datagram for a departed mobile node, it can then re-encapsulate the datagram and forward it to the mobile node (step 5) using the new COA. If the mobile node later moves yet again to a new foreign network, the foreign agent in that new visited network would then contact the anchor foreign agent in order to set up forwarding to this new foreign network.

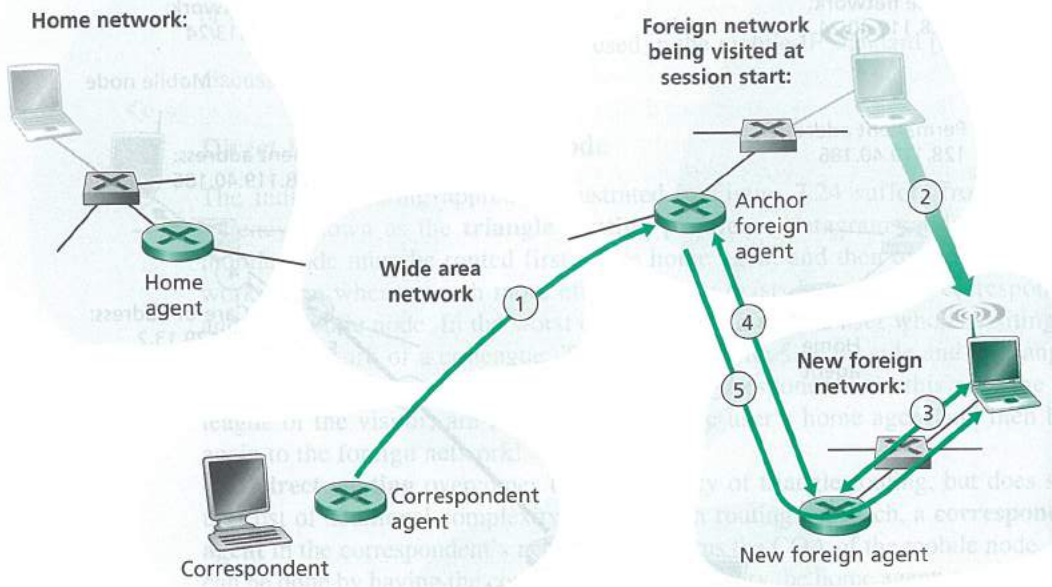


Figure 7.27 ♦ Mobile transfer between networks with direct routing

7.6 Mobile IP

The Internet architecture and protocols for supporting mobility, collectively known as mobile IP, are defined primarily in RFC 5944 for IPv4. Mobile IP is a flexible standard, supporting many different modes of operation (for example, operation with or without a foreign agent), multiple ways for agents and mobile nodes to discover each other, use of single or multiple COAs, and multiple forms of encapsulation. As such, mobile IP is a complex standard, and would require an entire book to describe in detail; indeed one such book is [Perkins 1998b]. Our modest goal here is to provide an overview of the most important aspects of mobile IP and to illustrate its use in a few common-case scenarios.

The mobile IP architecture contains many of the elements we have considered above, including the concepts of home agents, foreign agents, care-of addresses, and encapsulation/decapsulation. The current standard [RFC 5944] specifies the use of indirect routing to the mobile node.

The mobile IP standard consists of three main pieces:

- *Agent discovery.* Mobile IP defines the protocols used by a home or foreign agent to advertise its services to mobile nodes, and protocols for mobile nodes to solicit the services of a foreign or home agent.

- *Registration with the home agent.* Mobile IP defines the protocols used by the mobile node and/or foreign agent to register and deregister COAs with a mobile node's home agent.
- *Indirect routing of datagrams.* The standard also defines the manner in which datagrams are forwarded to mobile nodes by a home agent, including rules for forwarding datagrams, rules for handling error conditions, and several forms of encapsulation [RFC 2003, RFC 2004].

Security considerations are prominent throughout the mobile IP standard. For example, authentication of a mobile node is clearly needed to ensure that a malicious user does not register a bogus care-of address with a home agent, which could cause all datagrams addressed to an IP address to be redirected to the malicious user. Mobile IP achieves security using many of the mechanisms that we will examine in Chapter 8, so we will not address security considerations in our discussion below.

Agent Discovery

A mobile IP node arriving to a new network, whether attaching to a foreign network or returning to its home network, must learn the identity of the corresponding foreign or home agent. Indeed it is the discovery of a new foreign agent, with a new network address, that allows the network layer in a mobile node to learn that it has moved into a new foreign network. This process is known as **agent discovery**. Agent discovery can be accomplished in one of two ways: via agent advertisement or via agent solicitation.

With **agent advertisement**, a foreign or home agent advertises its services using an extension to the existing router discovery protocol [RFC 1256]. The agent periodically broadcasts an ICMP message with a type field of 9 (router discovery) on all links to which it is connected. The router discovery message contains the IP address of the router (that is, the agent), thus allowing a mobile node to learn the agent's IP address. The router discovery message also contains a mobility agent advertisement extension that contains additional information needed by the mobile node. Among the more important fields in the extension are the following:

- *Home agent bit (H).* Indicates that the agent is a home agent for the network in which it resides.
- *Foreign agent bit (F).* Indicates that the agent is a foreign agent for the network in which it resides.
- *Registration required bit (R).* Indicates that a mobile user in this network *must* register with a foreign agent. In particular, a mobile user cannot obtain a care-of address in the foreign network (for example, using DHCP) and assume the functionality of the foreign agent for itself, without registering with the foreign agent.

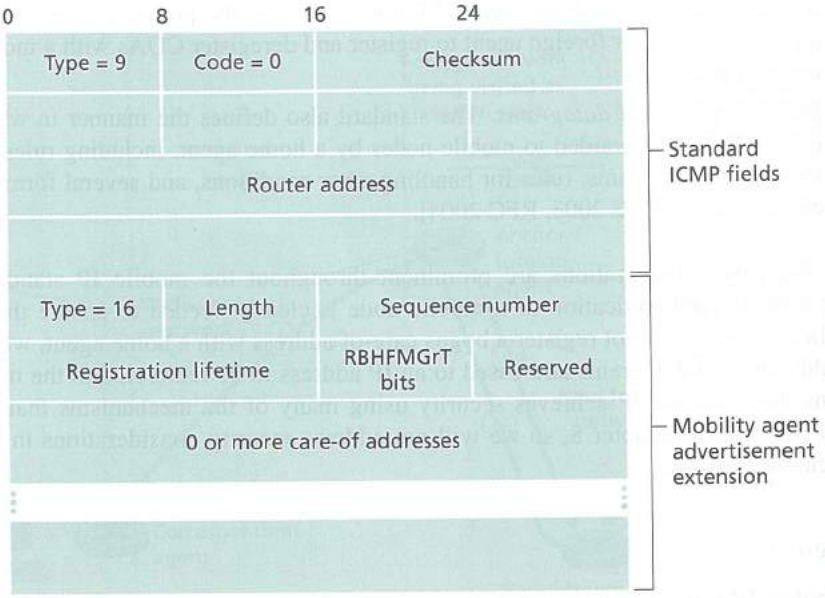


Figure 7.28 ♦ ICMP router discovery message with mobility agent advertisement extension

- *M, G encapsulation bits.* Indicate whether a form of encapsulation other than IP-in-IP encapsulation will be used.
- *Care-of address (COA) fields.* A list of one or more care-of addresses provided by the foreign agent. In our example below, the COA will be associated with the foreign agent, who will receive datagrams sent to the COA and then forward them to the appropriate mobile node. The mobile user will select one of these addresses as its COA when registering with its home agent.

Figure 7.28 illustrates some of the key fields in the agent advertisement message.

With **agent solicitation**, a mobile node wanting to learn about agents without waiting to receive an agent advertisement can broadcast an agent solicitation message, which is simply an ICMP message with type value 10. An agent receiving the solicitation will unicast an agent advertisement directly to the mobile node, which can then proceed as if it had received an unsolicited advertisement.

Registration with the Home Agent

Once a mobile IP node has received a COA, that address must be registered with the home agent. This can be done either via the foreign agent (who then registers the

COA with the home agent) or directly by the mobile IP node itself. We consider the former case below. Four steps are involved.

1. Following the receipt of a foreign agent advertisement, a mobile node sends a mobile IP registration message to the foreign agent. The registration message is carried within a UDP datagram and sent to port 434. The registration message carries a COA advertised by the foreign agent, the address of the home agent (HA), the permanent address of the mobile node (MA), the requested lifetime of the registration, and a 64-bit registration identification. The requested registration lifetime is the number of seconds that the registration is to be valid. If the registration is not renewed at the home agent within the specified lifetime, the registration will become invalid. The registration identifier acts like a sequence number and serves to match a received registration reply with a registration request, as discussed below.
2. The foreign agent receives the registration message and records the mobile node's permanent IP address. The foreign agent now knows that it should be looking for datagrams containing an encapsulated datagram whose destination address matches the permanent address of the mobile node. The foreign agent then sends a mobile IP registration message (again, within a UDP datagram) to port 434 of the home agent. The message contains the COA, HA, MA, encapsulation format requested, requested registration lifetime, and registration identification.
3. The home agent receives the registration request and checks for authenticity and correctness. The home agent binds the mobile node's permanent IP address with the COA; in the future, datagrams arriving at the home agent and addressed to the mobile node will now be encapsulated and tunneled to the COA. The home agent sends a mobile IP registration reply containing the HA, MA, actual registration lifetime, and the registration identification of the request that is being satisfied with this reply.
4. The foreign agent receives the registration reply and then forwards it to the mobile node.

At this point, registration is complete, and the mobile node can receive datagrams sent to its permanent address. Figure 7.29 illustrates these steps. Note that the home agent specifies a lifetime that is smaller than the lifetime requested by the mobile node.

A foreign agent need not explicitly deregister a COA when a mobile node leaves its network. This will occur automatically, when the mobile node moves to a new network (whether another foreign network or its home network) and registers a new COA.

The mobile IP standard allows many additional scenarios and capabilities in addition to those described previously. The interested reader should consult [Perkins 1998b; RFC 5944].

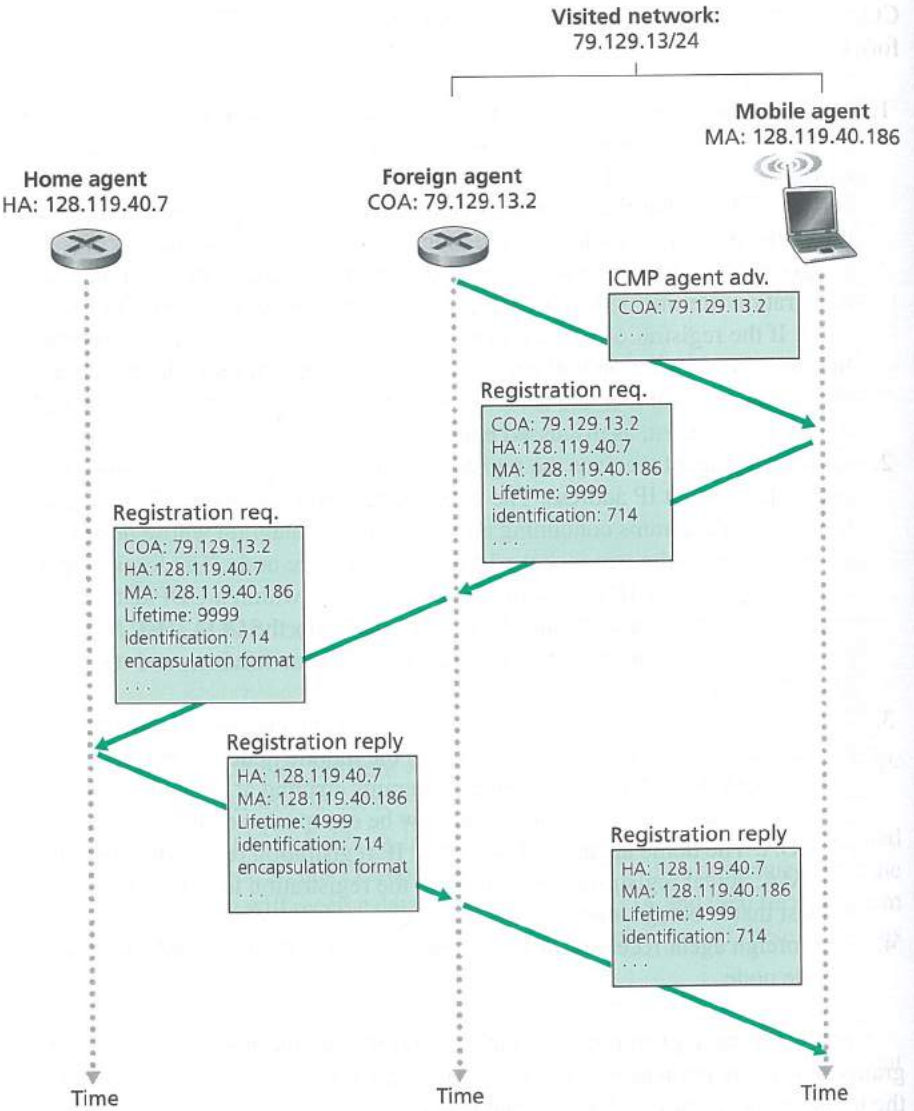


Figure 7.29 ♦ Agent advertisement and mobile IP registration

7.7 Managing Mobility in Cellular Networks

Having examined how mobility is managed in IP networks, let's now turn our attention to networks with an even longer history of supporting mobility—cellular telephony networks. Whereas we focused on the first-hop wireless link in cellular

networks in Section 7.4, we'll focus here on mobility, using the GSM cellular network [Goodman 1997; Mouly 1992; Scourias 2012; Kaaranen 2001; Korhonen 2003; Turner 2012] as our case study, since it is a mature and widely deployed technology. Mobility in 3G and 4G networks is similar in principle to that used in GSM. As in the case of mobile IP, we'll see that a number of the fundamental principles we identified in Section 7.5 are embodied in GSM's network architecture.

Like mobile IP, GSM adopts an indirect routing approach (see Section 7.5.2), first routing the correspondent's call to the mobile user's home network and from there to the visited network. In GSM terminology, the mobile users's home network is referred to as the mobile user's **home public land mobile network (home PLMN)**. Since the PLMN acronym is a bit of a mouthful, and mindful of our quest to avoid an alphabet soup of acronyms, we'll refer to the GSM home PLMN simply as the **home network**. The home network is the cellular provider with which the mobile user has a subscription (i.e., the provider that bills the user for monthly cellular service). The visited PLMN, which we'll refer to simply as the **visited network**, is the network in which the mobile user is currently residing.

As in the case of mobile IP, the responsibilities of the home and visited networks are quite different.

- The home network maintains a database known as the **home location register (HLR)**, which contains the permanent cell phone number and subscriber profile information for each of its subscribers. Importantly, the HLR also contains information about the current locations of these subscribers. That is, if a mobile user is currently roaming in another provider's cellular network, the HLR contains enough information to obtain (via a process we'll describe shortly) an address in the visited network to which a call to the mobile user should be routed. As we'll see, a special switch in the home network, known as the **Gateway Mobile services Switching Center (GMSC)** is contacted by a correspondent when a call is placed to a mobile user. Again, in our quest to avoid an alphabet soup of acronyms, we'll refer to the GMSC here by a more descriptive term, **home MSC**.
- The visited network maintains a database known as the **visitor location register (VLR)**. The VLR contains an entry for each mobile user that is *currently* in the portion of the network served by the VLR. VLR entries thus come and go as mobile users enter and leave the network. A VLR is usually co-located with the mobile switching center (MSC) that coordinates the setup of a call to and from the visited network.

In practice, a provider's cellular network will serve as a home network for its subscribers and as a visited network for mobile users whose subscription is with a different cellular provider.

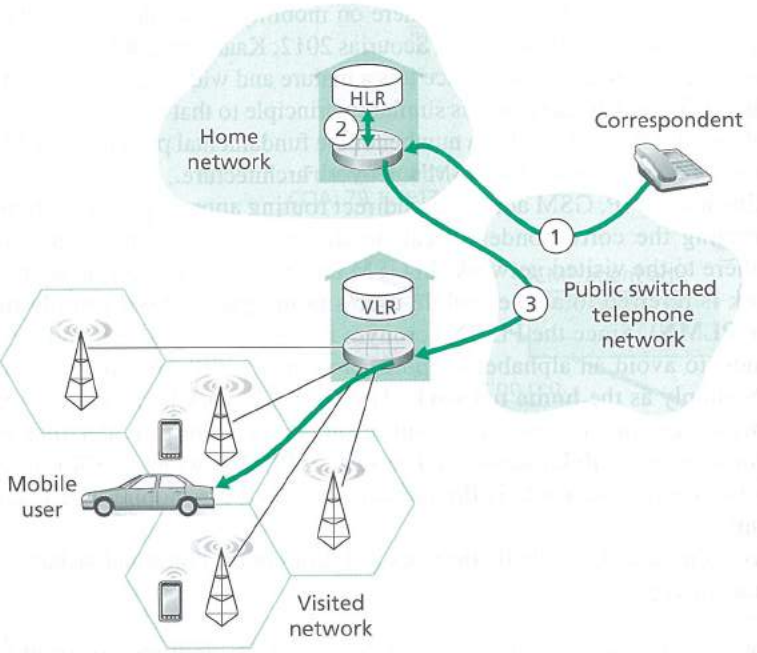


Figure 7.30 ♦ Placing a call to a mobile user: Indirect routing

7.7.1 Routing Calls to a Mobile User

We're now in a position to describe how a call is placed to a mobile GSM user in a visited network. We'll consider a simple example below; more complex scenarios are described in [Mouly 1992]. The steps, as illustrated in Figure 7.30, are as follows:

1. The correspondent dials the mobile user's phone number. This number itself does not refer to a particular telephone line or location (after all, the phone number is fixed and the user is mobile!). The leading digits in the number are sufficient to globally identify the mobile's home network. The call is routed from the correspondent through the PSTN to the home MSC in the mobile's home network. This is the first leg of the call.
2. The home MSC receives the call and interrogates the HLR to determine the location of the mobile user. In the simplest case, the HLR returns the **mobile station roaming number (MSRN)**, which we will refer to as the **roaming number**. Note that this number is different from the mobile's permanent phone number, which is associated with the mobile's home network. The roaming number is ephemeral: It is temporarily assigned to a mobile when it enters a visited network. The roaming number serves a role similar to that of the care-of

address in mobile IP and, like the COA, is invisible to the correspondent and the mobile. If HLR does not have the roaming number, it returns the address of the VLR in the visited network. In this case (not shown in Figure 7.30), the home MSC will need to query the VLR to obtain the roaming number of the mobile node. But how does the HLR get the roaming number or the VLR address in the first place? What happens to these values when the mobile user moves to another visited network? We'll consider these important questions shortly.

3. Given the roaming number, the home MSC sets up the second leg of the call through the network to the MSC in the visited network. The call is completed, being routed from the correspondent to the home MSC, and from there to the visited MSC, and from there to the base station serving the mobile user.

An unresolved question in step 2 is how the HLR obtains information about the location of the mobile user. When a mobile telephone is switched on or enters a part of a visited network that is covered by a new VLR, the mobile must register with the visited network. This is done through the exchange of signaling messages between the mobile and the VLR. The visited VLR, in turn, sends a location update request message to the mobile's HLR. This message informs the HLR of either the roaming number at which the mobile can be contacted, or the address of the VLR (which can then later be queried to obtain the mobile number). As part of this exchange, the VLR also obtains subscriber information from the HLR about the mobile and determines what services (if any) should be accorded the mobile user by the visited network.

7.7.2 Handoffs in GSM

A **handoff** occurs when a mobile station changes its association from one base station to another during a call. As shown in Figure 7.31, a mobile's call is initially (before handoff) routed to the mobile through one base station (which we'll refer to as the old base station), and after handoff is routed to the mobile through another base

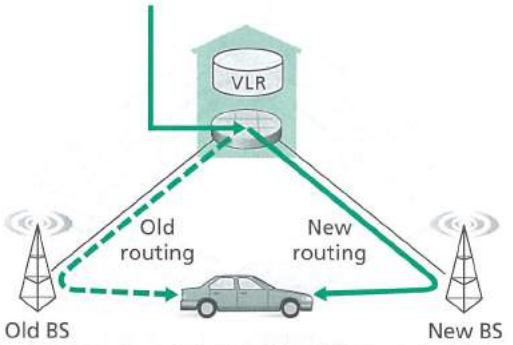


Figure 7.31 ♦ Handoff scenario between base stations with a common MSC

station (which we'll refer to as the new base station). Note that a handoff between base stations results not only in the mobile transmitting/receiving to/from a new base station, but also in the rerouting of the ongoing call from a switching point within the network to the new base station. Let's initially assume that the old and new base stations share the same MSC, and that the rerouting occurs at this MSC.

There may be several reasons for handoff to occur, including (1) the signal between the current base station and the mobile may have deteriorated to such an extent that the call is in danger of being dropped, and (2) a cell may have become overloaded, handling a large number of calls. This congestion may be alleviated by handing off mobiles to less congested nearby cells.

While it is associated with a base station, a mobile periodically measures the strength of a beacon signal from its current base station as well as beacon signals from nearby base stations that it can "hear." These measurements are reported once or twice a second to the mobile's current base station. Handoff in GSM is initiated by the old base station based on these measurements, the current loads of mobiles in nearby cells, and other factors [Mouly 1992]. The GSM standard does not specify the specific algorithm to be used by a base station to determine whether or not to perform handoff.

Figure 7.32 illustrates the steps involved when a base station does decide to hand off a mobile user:

1. The old base station (BS) informs the visited MSC that a handoff is to be performed and the BS (or possible set of BSs) to which the mobile is to be handed off.
2. The visited MSC initiates path setup to the new BS, allocating the resources needed to carry the rerouted call, and signaling the new BS that a handoff is about to occur.
3. The new BS allocates and activates a radio channel for use by the mobile.
4. The new BS signals back to the visited MSC and the old BS that the visited-MSC-to-new-BS path has been established and that the mobile should be

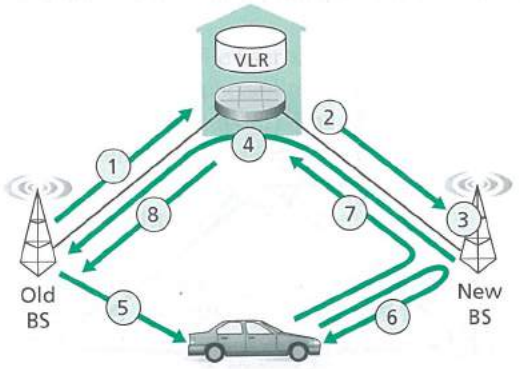
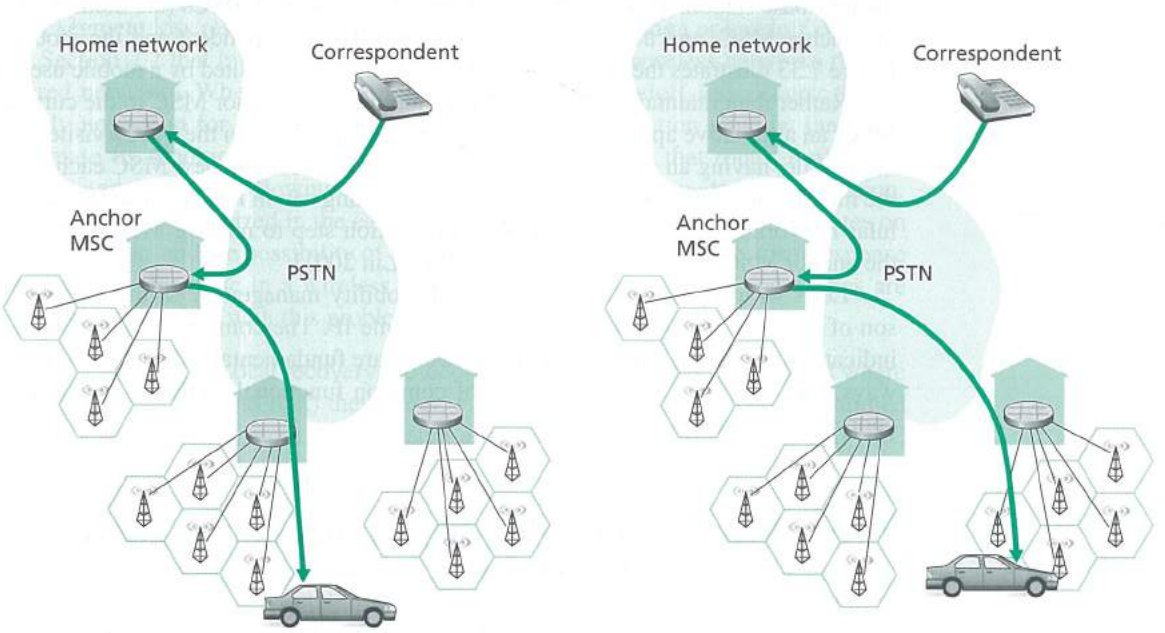


Figure 7.32 ♦ Steps in accomplishing a handoff between base stations with a common MSC

informed of the impending handoff. The new BS provides all of the information that the mobile will need to associate with the new BS.

5. The mobile is informed that it should perform a handoff. Note that up until this point, the mobile has been blissfully unaware that the network has been laying the groundwork (e.g., allocating a channel in the new BS and allocating a path from the visited MSC to the new BS) for a handoff.
6. The mobile and the new BS exchange one or more messages to fully activate the new channel in the new BS.
7. The mobile sends a handoff complete message to the new BS, which is forwarded up to the visited MSC. The visited MSC then reroutes the ongoing call to the mobile via the new BS.
8. The resources allocated along the path to the old BS are then released.

Let's conclude our discussion of handoff by considering what happens when the mobile moves to a BS that is associated with a *different* MSC than the old BS, and what happens when this inter-MSC handoff occurs more than once. As shown in Figure 7.33, GSM defines the notion of an **anchor MSC**. The anchor MSC is the MSC visited by the mobile when a call first begins; the anchor MSC thus remains unchanged during the call. Throughout the call's duration and regardless of the number of inter-MSC



a. Before handoff

b. After handoff

Figure 7.33 ♦ Rerouting via the anchor MSC

GSM element	Comment on GSM element	Mobile IP element
Home system	Network to which the mobile user's permanent phone number belongs.	Home network
Gateway mobile switching center or simply home MSC, Home location register (HLR)	Home MSC: point of contact to obtain routable address of mobile user. HLR: database in home system containing permanent phone number, profile information, current location of mobile user, subscription information.	Home agent
Visited system	Network other than home system where mobile user is currently residing.	Visited network
Visited mobile services switching center, Visitor location register (VLR)	Visited MSC: responsible for setting up calls to/from mobile nodes in cells associated with MSC. VLR: temporary database entry in visited system, containing subscription information for each visiting mobile user.	Foreign agent
Mobile station roaming number (MSRN) or simply roaming number	Routable address for telephone call segment between home MSC and visited MSC, visible to neither the mobile nor the correspondent.	Care-of address

Table 7.2 ♦ Commonalities between mobile IP and GSM mobility

transfers performed by the mobile, the call is routed from the home MSC to the anchor MSC, and then from the anchor MSC to the visited MSC where the mobile is currently located. When a mobile moves from the coverage area of one MSC to another, the ongoing call is rerouted from the anchor MSC to the new visited MSC containing the new base station. Thus, at all times there are at most three MSCs (the home MSC, the anchor MSC, and the visited MSC) between the correspondent and the mobile. Figure 7.33 illustrates the routing of a call among the MSCs visited by a mobile user.

Rather than maintaining a single MSC hop from the anchor MSC to the current MSC, an alternative approach would have been to simply chain the MSCs visited by the mobile, having an old MSC forward the ongoing call to the new MSC each time the mobile moves to a new MSC. Such MSC chaining can in fact occur in IS-41 cellular networks, with an optional path minimization step to remove MSCs between the anchor MSC and the current visited MSC [Lin 2001].

Let's wrap up our discussion of GSM mobility management with a comparison of mobility management in GSM and Mobile IP. The comparison in Table 7.2 indicates that although IP and cellular networks are fundamentally different in many ways, they share a surprising number of common functional elements and overall approaches in handling mobility.

7.8 Wireless and Mobility: Impact on Higher-Layer Protocols

In this chapter, we've seen that wireless networks differ significantly from their wired counterparts at both the link layer (as a result of wireless channel characteristics such as fading, multipath, and hidden terminals) and at the network layer

(as a result of mobile users who change their points of attachment to the network). But are there important differences at the transport and application layers? It's tempting to think that these differences will be minor, since the network layer provides the same best-effort delivery service model to upper layers in both wired and wireless networks. Similarly, if protocols such as TCP or UDP are used to provide transport-layer services to applications in both wired and wireless networks, then the application layer should remain unchanged as well. In one sense our intuition is right—TCP and UDP can (and do) operate in networks with wireless links. On the other hand, transport protocols in general, and TCP in particular, can sometimes have very different performance in wired and wireless networks, and it is here, in terms of performance, that differences are manifested. Let's see why.

Recall that TCP retransmits a segment that is either lost or corrupted on the path between sender and receiver. In the case of mobile users, loss can result from either network congestion (router buffer overflow) or from handoff (e.g., from delays in rerouting segments to a mobile's new point of attachment to the network). In all cases, TCP's receiver-to-sender ACK indicates only that a segment was not received intact; the sender is unaware of whether the segment was lost due to congestion, during handoff, or due to detected bit errors. In all cases, the sender's response is the same—to retransmit the segment. TCP's congestion-control response is *also* the same in all cases—TCP decreases its congestion window, as discussed in Section 3.7. By unconditionally decreasing its congestion window, TCP implicitly assumes that segment loss results from congestion rather than corruption or handoff. We saw in Section 7.2 that bit errors are much more common in wireless networks than in wired networks. When such bit errors occur or when handoff loss occurs, there's really no reason for the TCP sender to decrease its congestion window (and thus decrease its sending rate). Indeed, it may well be the case that router buffers are empty and packets are flowing along the end-to-end path unimpeded by congestion.

Researchers realized in the early to mid 1990s that given high bit error rates on wireless links and the possibility of handoff loss, TCP's congestion-control response could be problematic in a wireless setting. Three broad classes of approaches are possible for dealing with this problem:

- *Local recovery.* Local recovery protocols recover from bit errors when and where (e.g., at the wireless link) they occur, e.g., the 802.11 ARQ protocol we studied in Section 7.3, or more sophisticated approaches that use both ARQ and FEC [Ayanoglu 1995].
- *TCP sender awareness of wireless links.* In the local recovery approaches, the TCP sender is blissfully unaware that its segments are traversing a wireless link. An alternative approach is for the TCP sender and receiver to be aware of the existence of a wireless link, to distinguish between congestive losses occurring in the wired network and corruption/loss occurring at the wireless link, and to invoke congestion control only in response to congestive wired-network losses. [Balakrishnan 1997] investigates various types of TCP, assuming that end systems

can make this distinction. [Liu 2003] investigates techniques for distinguishing between losses on the wired and wireless segments of an end-to-end path.

- *Split-connection approaches.* In a split-connection approach [Bakre 1995], the end-to-end connection between the mobile user and the other end point is broken into two transport-layer connections: one from the mobile host to the wireless access point, and one from the wireless access point to the other communication end point (which we'll assume here is a wired host). The end-to-end connection is thus formed by the concatenation of a wireless part and a wired part. The transport layer over the wireless segment can be a standard TCP connection [Bakre 1995], or a specially tailored error recovery protocol on top of UDP. [Yavatkar 1994] investigates the use of a transport-layer selective repeat protocol over the wireless connection. Measurements reported in [Wei 2006] indicate that split TCP connections are widely used in cellular data networks, and that significant improvements can indeed be made through the use of split TCP connections.

Our treatment of TCP over wireless links has been necessarily brief here. In-depth surveys of TCP challenges and solutions in wireless networks can be found in [Hanabali 2005; Leung 2006]. We encourage you to consult the references for details of this ongoing area of research.

Having considered transport-layer protocols, let us next consider the effect of wireless and mobility on application-layer protocols. Here, an important consideration is that wireless links often have relatively low bandwidths, as we saw in Figure 7.2. As a result, applications that operate over wireless links, particularly over cellular wireless links, must treat bandwidth as a scarce commodity. For example, a Web server serving content to a Web browser executing on a 4G phone will likely not be able to provide the same image-rich content that it gives to a browser operating over a wired connection. Although wireless links do provide challenges at the application layer, the mobility they enable also makes possible a rich set of location-aware and context-aware applications [Chen 2000; Baldauf 2007]. More generally, wireless and mobile networks will play a key role in realizing the ubiquitous computing environments of the future [Weiser 1991]. It's fair to say that we've only seen the tip of the iceberg when it comes to the impact of wireless and mobile networks on networked applications and their protocols!

7.9 Summary

Wireless and mobile networks have revolutionized telephony and are having an increasingly profound impact in the world of computer networks as well. With their anytime, anywhere, untethered access into the global network infrastructure, they are not only making network access more ubiquitous, they are also enabling an exciting new set of location-dependent services. Given the growing importance of wireless and

mobile networks, this chapter has focused on the principles, common link technologies, and network architectures for supporting wireless and mobile communication.

We began this chapter with an introduction to wireless and mobile networks, drawing an important distinction between the challenges posed by the *wireless* nature of the communication links in such networks, and by the *mobility* that these wireless links enable. This allowed us to better isolate, identify, and master the key concepts in each area. We focused first on wireless communication, considering the characteristics of a wireless link in Section 7.2. In Sections 7.3 and 7.4, we examined the link-level aspects of the IEEE 802.11 (WiFi) wireless LAN standard, two IEEE 802.15 personal area networks (Bluetooth and Zigbee), and 3G and 4G cellular Internet access. We then turned our attention to the issue of mobility. In Section 7.5, we identified several forms of mobility, with points along this spectrum posing different challenges and admitting different solutions. We considered the problems of locating and routing to a mobile user, as well as approaches for handing off the mobile user who dynamically moves from one point of attachment to the network to another. We examined how these issues were addressed in the mobile IP standard and in GSM, in Sections 7.6 and 7.7, respectively. Finally, we considered the impact of wireless links and mobility on transport-layer protocols and networked applications in Section 7.8.

Although we have devoted an entire chapter to the study of wireless and mobile networks, an entire book (or more) would be required to fully explore this exciting and rapidly expanding field. We encourage you to delve more deeply into this field by consulting the many references provided in this chapter.

Homework Problems and Questions

Chapter 7 Review Questions

SECTION 7.1

R1. What does it mean for a wireless network to be operating in "infrastructure mode"? If the network is not in infrastructure mode, what mode of operation is it in, and what is the difference between that mode of operation and infrastructure mode?

R2. What are the four types of wireless networks identified in our taxonomy in Section 7.1? Which of these types of wireless networks have you used?

SECTION 7.2

R3. What are the differences between the following types of wireless channel impairments: path loss, multipath propagation, interference from other sources?

R4. As a mobile node gets farther and farther away from a base station, what are two actions that a base station could take to ensure that the loss probability of a transmitted frame does not increase?

SECTIONS 7.3 AND 7.4

R5. Describe the role of the beacon frames in 802.11.

R6. True or false: Before an 802.11 station transmits a data frame, it must first send an RTS frame and receive a corresponding CTS frame.

R7. Why are acknowledgments used in 802.11 but not in wired Ethernet?

R8. True or false: Ethernet and 802.11 use the same frame structure.

R9. Describe how the RTS threshold works.

R10. Suppose the IEEE 802.11 RTS and CTS frames were as long as the standard DATA and ACK frames. Would there be any advantage to using the CTS and RTS frames? Why or why not?

R11. Section 7.3.4 discusses 802.11 mobility, in which a wireless station moves from one BSS to another within the same subnet. When the APs are interconnected with a switch, an AP may need to send a frame with a spoofed MAC address to get the switch to forward the frame properly. Why?

R12. What are the differences between a master device in a Bluetooth network and a base station in an 802.11 network?

R13. What is meant by a super frame in the 802.15.4 Zigbee standard?

R14. What is the role of the "core network" in the 3G cellular data architecture?

R15. What is the role of the RNC in the 3G cellular data network architecture? What role does the RNC play in the cellular voice network?

R16. What is the role of the eNodeB, MME, P-GW, and S-GW in 4G architecture?

R17. What are three important differences between the 3G and 4G cellular architectures?

SECTIONS 7.5 AND 7.6

R18. If a node has a wireless connection to the Internet, does that node have to be mobile? Explain. Suppose that a user with a laptop walks around her house with her laptop, and always accesses the Internet through the same access point. Is this user mobile from a network standpoint? Explain.

R19. What is the difference between a permanent address and a care-of address? Who assigns a care-of address?

R20. Consider a TCP connection going over Mobile IP. True or false: The TCP connection phase between the correspondent and the mobile host goes through the mobile's home network, but the data transfer phase is directly between the correspondent and the mobile host, bypassing the home network.

SECTION 7.7

R21. What are the purposes of the HLR and VLR in GSM networks? What elements of mobile IP are similar to the HLR and VLR?

R22. What is the role of the anchor MSC in GSM networks?

SECTION 7.8

R23. What are three approaches that can be taken to avoid having a single wireless link degrade the performance of an end-to-end transport-layer TCP connection?

Problems

P1. Consider the single-sender CDMA example in Figure 7.5. What would be the sender's output (for the 2 data bits shown) if the sender's CDMA code were (1, −1, 1, −1, 1, −1, 1, −1)?

P2. Consider sender 2 in Figure 7.6. What is the sender's output to the channel (before it is added to the signal from sender 1), $Z^2_{i,m}$?

P3. Suppose that the receiver in Figure 7.6 wanted to receive the data being sent by sender 2. Show (by calculation) that the receiver is indeed able to recover sender 2's data from the aggregate channel signal by using sender 2's code.

P4. For the two-sender, two-receiver example, give an example of two CDMA codes containing 1 and 21 values that do not allow the two receivers to extract the original transmitted bits from the two CDMA senders.

P5. Suppose there are two ISPs providing WiFi access in a particular café, with each ISP operating its own AP and having its own IP address block.

a. Further suppose that by accident, each ISP has configured its AP to operate over channel 11. Will the 802.11 protocol completely break down in this situation? Discuss what happens when two stations, each associated with a different ISP, attempt to transmit at the same time.

b. Now suppose that one AP operates over channel 1 and the other over channel 11. How do your answers change?

P6. In step 4 of the CSMA/CA protocol, a station that successfully transmits a frame begins the CSMA/CA protocol for a second frame at step 2, rather than at step 1. What rationale might the designers of CSMA/CA have had in mind by having such a station not transmit the second frame immediately (if the channel is sensed idle)?

P7. Suppose an 802.11b station is configured to always reserve the channel with the RTS/CTS sequence. Suppose this station suddenly wants to transmit 1,000 bytes of data, and all other stations are idle at this time. As a function of SIFS and DIFS, and ignoring propagation delay and assuming no bit errors, calculate the time required to transmit the frame and receive the acknowledgment.

P8. Consider the scenario shown in Figure 7.34, in which there are four wireless nodes, A, B, C, and D. The radio coverage of the four nodes is shown via the shaded ovals; all nodes share the same frequency. When A transmits, it

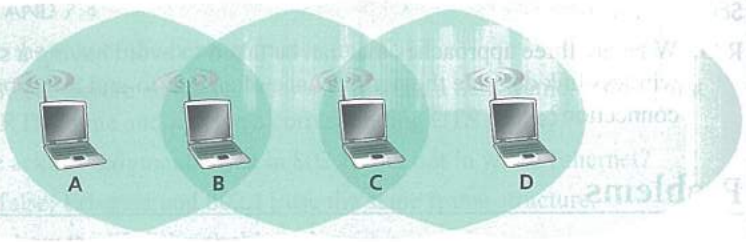


Figure 7.34 ♦ Scenario for problem P8

can only be heard/received by B; when B transmits, both A and C can hear/receive from B; when C transmits, both B and D can hear/receive from C; when D transmits, only C can hear/receive from D.

Suppose now that each node has an infinite supply of messages that it wants to send to each of the other nodes. If a message's destination is not an immediate neighbor, then the message must be relayed. For example, if A wants to send to D, a message from A must first be sent to B, which then sends the message to C, which then sends the message to D. Time is slotted, with a message transmission time taking exactly one time slot, e.g., as in slotted Aloha. During a slot, a node can do one of the following: (*i*) send a message, (*ii*) receive a message (if exactly one message is being sent to it), (*iii*) remain silent. As always, if a node hears two or more simultaneous transmissions, a collision occurs and none of the transmitted messages are received successfully. You can assume here that there are no bit-level errors, and thus if exactly one message is sent, it will be received correctly by those within the transmission radius of the sender.

a. Suppose now that an omniscient controller (i.e., a controller that knows the state of every node in the network) can command each node to do whatever it (the omniscient controller) wishes, i.e., to send a message, to receive a message, or to remain silent. Given this omniscient controller, what is the maximum rate at which a data message can be transferred from C to A, given that there are no other messages between any other source/destination pairs?

b. Suppose now that A sends messages to B, and D sends messages to C. What is the combined maximum rate at which data messages can flow from A to B and from D to C?

c. Suppose now that A sends messages to B, and C sends messages to D. What is the combined maximum rate at which data messages can flow from A to B and from C to D?

d. Suppose now that the wireless links are replaced by wired links. Repeat questions (a) through (c) again in this wired scenario.

e. Now suppose we are again in the wireless scenario, and that for every data message sent from source to destination, the destination will send an ACK message back to the source (e.g., as in TCP). Also suppose that each ACK message takes up one slot. Repeat questions (a)–(c) above for this scenario.

P9. Describe the format of the 802.15.1 Bluetooth frame. You will have to do some reading outside of the text to find this information. Is there anything in the frame format that inherently limits the number of active nodes in an 802.15.1 network to eight active nodes? Explain.

P10. Consider the following idealized LTE scenario. The downstream channel (see Figure 7.21) is slotted in time, across F frequencies. There are four nodes, A, B, C, and D, reachable from the base station at rates of 10 Mbps, 5 Mbps, 2.5 Mbps, and 1 Mbps, respectively, on the downstream channel. These rates assume that the base station utilizes all time slots available on all F frequencies to send to just one station. The base station has an infinite amount of data to send to each of the nodes, and can send to any one of these four nodes using any of the F frequencies during any time slot in the downstream sub-frame.

a. What is the maximum rate at which the base station can send to the nodes, assuming it can send to any node it chooses during each time slot? Is your solution fair? Explain and define what you mean by "fair."

b. If there is a fairness requirement that each node must receive an equal amount of data during each one second interval, what is the average transmission rate by the base station (to all nodes) during the downstream sub-frame? Explain how you arrived at your answer.

c. Suppose that the fairness criterion is that any node can receive at most twice as much data as any other node during the sub-frame. What is the average transmission rate by the base station (to all nodes) during the sub-frame? Explain how you arrived at your answer.

P11. In Section 7.5, one proposed solution that allowed mobile users to maintain their IP addresses as they moved among foreign networks was to have a foreign network advertise a highly specific route to the mobile user and use the existing routing infrastructure to propagate this information throughout the network. We identified scalability as one concern. Suppose that when a mobile user moves from one network to another, the new foreign network advertises a specific route to the mobile user, and the old foreign network withdraws its route. Consider how routing information propagates in a distance-vector algorithm (particularly for the case of interdomain routing among networks that span the globe).

a. Will other routers be able to route datagrams immediately to the new foreign network as soon as the foreign network begins advertising its route?

b. Is it possible for different routers to believe that different foreign networks contain the mobile user?

c. Discuss the timescale over which other routers in the network will eventually learn the path to the mobile users.

P12. Suppose the correspondent in Figure 7.23 were mobile. Sketch the additional network-layer infrastructure that would be needed to route the datagram from the original mobile user to the (now mobile) correspondent. Show the structure of the datagram(s) between the original mobile user and the (now mobile) correspondent, as in Figure 7.24.

P13. In mobile IP, what effect will mobility have on end-to-end delays of datagrams between the source and destination?

P14. Consider the chaining example discussed at the end of Section 7.7.2. Suppose a mobile user visits foreign networks A, B, and C, and that a correspondent begins a connection to the mobile user when it is resident in foreign network A. List the sequence of messages between foreign agents, and between foreign agents and the home agent as the mobile user moves from network A to network B to network C. Next, suppose chaining is not performed, and the correspondent (as well as the home agent) must be explicitly notified of the changes in the mobile user's care-of address. List the sequence of messages that would need to be exchanged in this second scenario.

P15. Consider two mobile nodes in a foreign network having a foreign agent. Is it possible for the two mobile nodes to use the same care-of address in mobile IP? Explain your answer.

P16. In our discussion of how the VLR updated the HLR with information about the mobile's current location, what are the advantages and disadvantages of providing the MSRN as opposed to the address of the VLR to the HLR?

Wireshark Lab

At the Web site for this textbook, www.pearsonhighered.com/cs-resources, you'll find a Wireshark lab for this chapter that captures and studies the 802.11 frames exchanged between a wireless laptop and an access point.

Deborah Estrin

Deborah Estrin is a Professor of Computer Science at Cornell Tech in New York City and a Professor of Public Health at Weill Cornell Medical College. She is founder of the **Health Tech Hub** at Cornell Tech and co-founder of the non-profit startup **Open mHealth**. She received her Ph.D. (1985) in Computer Science from M.I.T. and her B.S. (1980) from UC Berkeley. Estrin's early research focused on the design of network protocols, including multicast and inter-domain routing. In 2002 Estrin founded the NSF-funded Science and Technology Center at UCLA, Center for Embedded Networked Sensing (CENS http://cens.ucla.edu.). CENS launched new areas of multi-disciplinary computer systems research from sensor networks for environmental monitoring, to participatory sensing for citizen science. Her current focus is on mobile health and small data, leveraging the pervasiveness of mobile devices and digital interactions for health and life management, as described in her 2013 TEDMED talk. Professor Estrin is an elected member of the American Academy of Arts and Sciences (2007) and the National Academy of Engineering (2009). She is a fellow of the IEEE, ACM, and AAAS. She was selected as the first ACM-W Athena Lecturer (2006), awarded the Anita Borg Institute's Women of Vision Award for Innovation (2007), inducted into the WITI hall of fame (2008) and awarded *Doctor Honoris Causa* from EPFL (2008) and Uppsala University (2011).

Please describe a few of the most exciting projects you have worked on during your career. What were the biggest challenges?

In the mid-90s at USC and ISI, I had the great fortune to work with the likes of Steve Deering, Mark Handley, and Van Jacobson on the design of multicast routing protocols (in particular, PIM). I tried to carry many of the architectural design lessons from multicast into the design of ecological monitoring arrays, where for the first time I really began to take applications and multidisciplinary research seriously. That interest in jointly innovating in the social and technological space is what interests me so much about my latest area of research, mobile health. The challenges in these projects were as diverse as the problem

domains, but what they all had in common was the need to keep our eyes open to whether we had the problem definition right as we iterated between design and deployment, prototype and pilot. None of them were problems that could be solved analytically, with simulation or even in constructed laboratory experiments. They all challenged our ability to retain clean architectures in the presence of messy problems and contexts, and they all called for extensive collaboration.

What changes and innovations do you see happening in wireless networks and mobility in the future?

In a prior edition of this interview I said that I have never put much faith into predicting the future, but I did go on to speculate that we might see the end of feature phones (i.e., those that are not programmable and are used only for voice and text messaging) as smart phones become more and more powerful and the primary point of Internet access for many—and now not so many years later that is clearly the case. I also predicted that we would see the continued proliferation of embedded SIMs by which all sorts of devices have the ability to communicate via the cellular network at low data rates. While that has occurred, we see many devices and "Internet of Things" that use embedded WiFi and other lower power, shorter range, forms of connectivity to local hubs. I did not anticipate at that time the emergence of a large consumer wearables market. By the time the next edition is published I expect broad proliferation of personal applications that leverage data from IoT and other digital traces.

Where do you see the future of networking and the Internet?

Again I think its useful to look both back and forward. Previously I observed that the efforts in named data and software-defined networking would emerge to create a more manageable, evolvable, and richer infrastructure and more generally represent moving the role of architecture higher up in the stack. In the beginnings of the Internet, architecture was layer 4 and below, with applications being more siloed/monolithic, sitting on top. Now data and analytics dominate transport. The adoption of SDN (which I'm really happy to see is featured in this 7th edition of this book) has been well beyond what I ever anticipated. However, looking up the stack, our dominant applications increasingly live in walled gardens, whether mobile apps or large consumer platforms such as Facebook. As Data Science and Big Data techniques develop, they might help to lure these applications out of their silos because of the value in connecting with other apps and platforms.

What people inspired you professionally?

There are three people who come to mind. First, Dave Clark, the secret sauce and undersung hero of the Internet community. I was lucky to be around in the early days to see him act as the "organizing principle" of the IAB and Internet governance; the priest of rough consensus and running code. Second, Scott Shenker, for his intellectual brilliance, integrity, and persistence. I strive for, but rarely attain, his clarity in defining problems and solutions. He is always the first person I e-mail for advice on matters large and small. Third, my sister Judy Estrin, who had the creativity and courage to spend her career bringing ideas and concepts to market. Without the Judys of the world the Internet technologies would never have transformed our lives.

What are your recommendations for students who want careers in computer science and networking?

First, build a strong foundation in your academic work, balanced with any and every real-world work experience you can get. As you look for a working environment, seek opportunities in problem areas you really care about and with smart teams that you can learn from.

CHAPTER 8

Security in Computer Networks

Way back in Section 1.6 we described some of the more prevalent and damaging classes of Internet attacks, including malware attacks, denial of service, sniffing, source masquerading, and message modification and deletion. Although we have since learned a tremendous amount about computer networks, we still haven't examined how to secure networks from those attacks. Equipped with our newly acquired expertise in computer networking and Internet protocols, we'll now study in-depth secure communication and, in particular, how computer networks can be defended from those nasty bad guys.

Let us introduce Alice and Bob, two people who want to communicate and wish to do so "securely." This being a networking text, we should remark that Alice and Bob could be two routers that want to exchange routing tables securely, a client and server that want to establish a secure transport connection, or two e-mail applications that want to exchange secure e-mail—all case studies that we will consider later in this chapter. Alice and Bob are well-known fixtures in the security community, perhaps because their names are more fun than a generic entity named "A" that wants to communicate securely with a generic entity named "B." Love affairs, wartime communication, and business transactions are the commonly cited human needs for secure communications; preferring the first to the latter two, we're happy to use Alice and Bob as our sender and receiver, and imagine them in this first scenario.

We said that Alice and Bob want to communicate and wish to do so "securely," but what precisely does this mean? As we will see, security (like love) is a many-splendored thing; that is, there are many facets to security. Certainly, Alice and Bob would like for the contents of their communication to remain secret from an eavesdropper. They probably would also like to make sure that when they are

communicating, they are indeed communicating with each other, and that if their communication is tampered with by an eavesdropper, that this tampering is detected. In the first part of this chapter, we'll cover the fundamental cryptography techniques that allow for encrypting communication, authenticating the party with whom one is communicating, and ensuring message integrity.

In the second part of this chapter, we'll examine how the fundamental cryptography principles can be used to create secure networking protocols. Once again taking a top-down approach, we'll examine secure protocols in each of the (top four) layers, beginning with the application layer. We'll examine how to secure e-mail, how to secure a TCP connection, how to provide blanket security at the network layer, and how to secure a wireless LAN. In the third part of this chapter we'll consider operational security, which is about protecting organizational networks from attacks. In particular, we'll take a careful look at how firewalls and intrusion detection systems can enhance the security of an organizational network.

8.1 What Is Network Security?

Let's begin our study of network security by returning to our lovers, Alice and Bob, who want to communicate "securely." What precisely does this mean? Certainly, Alice wants only Bob to be able to understand a message that she has sent, even though they *are* communicating over an insecure medium where an intruder (Trudy, the intruder) may intercept whatever is transmitted from Alice to Bob. Bob also wants to be sure that the message he receives from Alice was indeed sent by Alice, and Alice wants to make sure that the person with whom she is communicating is indeed Bob. Alice and Bob also want to make sure that the contents of their messages have not been altered in transit. They also want to be assured that they can communicate in the first place (i.e., that no one denies them access to the resources needed to communicate). Given these considerations, we can identify the following desirable properties of **secure communication**.

- *Confidentiality.* Only the sender and intended receiver should be able to understand the contents of the transmitted message. Because eavesdroppers may intercept the message, this necessarily requires that the message be somehow **encrypted** so that an intercepted message cannot be understood by an interceptor. This aspect of confidentiality is probably the most commonly perceived meaning of the term *secure communication*. We'll study cryptographic techniques for encrypting and decrypting data in Section 8.2.
- *Message integrity.* Alice and Bob want to ensure that the content of their communication is not altered, either maliciously or by accident, in transit. Extensions to the checksumming techniques that we encountered in reliable transport

and data link protocols can be used to provide such message integrity. We will study message integrity in Section 8.3.

- *End-point authentication.* Both the sender and receiver should be able to confirm the identity of the other party involved in the communication—to confirm that the other party is indeed who or what they claim to be. Face-to-face human communication solves this problem easily by visual recognition. When communicating entities exchange messages over a medium where they cannot see the other party, authentication is not so simple. When a user wants to access an inbox, how does the mail server verify that the user is the person he or she claims to be? We study end-point authentication in Section 8.4.
- *Operational security.* Almost all organizations (companies, universities, and so on) today have networks that are attached to the public Internet. These networks therefore can potentially be compromised. Attackers can attempt to deposit worms into the hosts in the network, obtain corporate secrets, map the internal network configurations, and launch DoS attacks. We'll see in Section 8.9 that operational devices such as firewalls and intrusion detection systems are used to counter attacks against an organization's network. A firewall sits between the organization's network and the public network, controlling packet access to and from the network. An intrusion detection system performs "deep packet inspection," alerting the network administrators about suspicious activity.

Having established what we mean by network security, let's next consider exactly what information an intruder may have access to, and what actions can be taken by the intruder. Figure 8.1 illustrates the scenario. Alice, the sender, wants to send data to Bob, the receiver. In order to exchange data securely, while meeting the requirements of confidentiality, end-point authentication, and message integrity, Alice and Bob will exchange control messages and data messages (in much the same way that TCP senders and receivers exchange control segments and data segments).

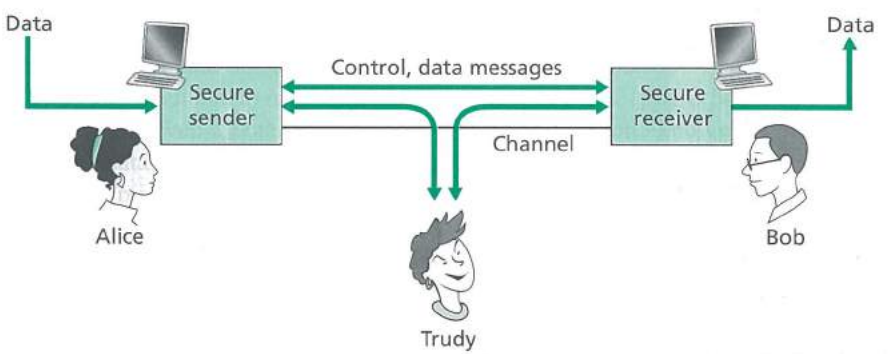


Figure 8.1 ♦ Sender, receiver, and intruder (Alice, Bob, and Trudy)

All or some of these messages will typically be encrypted. As discussed in Section 1.6, an intruder can potentially perform

- *eavesdropping*—sniffing and recording control and data messages on the channel.
- *modification, insertion*, or *deletion* of messages or message content.

As we'll see, unless appropriate countermeasures are taken, these capabilities allow an intruder to mount a wide variety of security attacks: snooping on communication (possibly stealing passwords and data), impersonating another entity, hijacking an ongoing session, denying service to legitimate network users by overloading system resources, and so on. A summary of reported attacks is maintained at the CERT Coordination Center [CERT 2016].

Having established that there are indeed real threats loose in the Internet, what are the Internet equivalents of Alice and Bob, our friends who need to communicate securely? Certainly, Bob and Alice might be human users at two end systems, for example, a real Alice and a real Bob who really do want to exchange secure e-mail. They might also be participants in an electronic commerce transaction. For example, a real Bob might want to transfer his credit card number securely to a Web server to purchase an item online. Similarly, a real Alice might want to interact with her bank online. The parties needing secure communication might themselves also be part of the network infrastructure. Recall that the domain name system (DNS, see Section 2.4) or routing daemons that exchange routing information (see Chapter 5) require secure communication between two parties. The same is true for network management applications, a topic we examined in Chapter 5). An intruder that could actively interfere with DNS lookups (as discussed in Section 2.4), routing computations [RFC 4272], or network management functions [RFC 3414] could wreak havoc in the Internet.

Having now established the framework, a few of the most important definitions, and the need for network security, let us next delve into cryptography. While the use of cryptography in providing confidentiality is self-evident, we'll see shortly that it is also central to providing end-point authentication and message integrity—making cryptography a cornerstone of network security.

8.2 Principles of Cryptography

Although cryptography has a long history dating back at least as far as Julius Caesar, modern cryptographic techniques, including many of those used in the Internet, are based on advances made in the past 30 years. Kahn's book, *The Codebreakers* [Kahn 1967], and Singh's book, *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography* [Singh 1999], provide a fascinating look at the

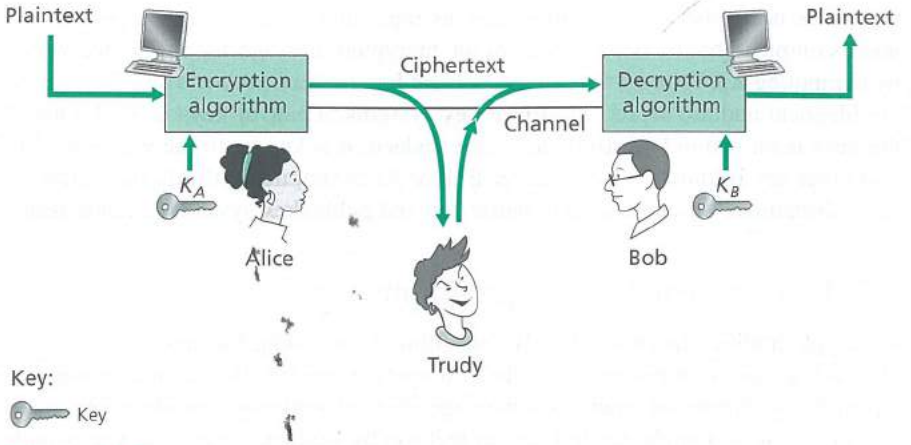


Figure 8.2 ♦ Cryptographic components

long history of cryptography. A complete discussion of cryptography itself requires a complete book [Kaufman 1995; Schneier 1995] and so we only touch on the essential aspects of cryptography, particularly as they are practiced on the Internet. We also note that while our focus in this section will be on the use of cryptography for confidentiality, we'll see shortly that cryptographic techniques are inextricably woven into authentication, message integrity, nonrepudiation, and more.

Cryptographic techniques allow a sender to disguise data so that an intruder can gain no information from the intercepted data. The receiver, of course, must be able to recover the original data from the disguised data. Figure 8.2 illustrates some of the important terminology.

Suppose now that Alice wants to send a message to Bob. Alice's message in its original form (for example, "`Bob, I love you. Alice`") is known as **plaintext**, or **cleartext**. Alice encrypts her plaintext message using an **encryption algorithm** so that the encrypted message, known as **ciphertext**, looks unintelligible to any intruder. Interestingly, in many modern cryptographic systems, including those used in the Internet, the encryption technique itself is *known*—published, standardized, and available to everyone (for example, [RFC 1321; RFC 3447; RFC 2420; NIST 2001]), even a potential intruder! Clearly, if everyone knows the method for encoding data, then there must be some secret information that prevents an intruder from decrypting the transmitted data. This is where keys come in.

In Figure 8.2, Alice provides a **key**, K_A, a string of numbers or characters, as input to the encryption algorithm. The encryption algorithm takes the key and the plaintext message, m, as input and produces ciphertext as output. The notation $K_A(m)$ refers to the ciphertext form (encrypted using the key K_A) of the plaintext message, m. The actual encryption algorithm that uses key K_A will be evident from the context. Similarly, Bob will provide a key, K_B, to the **decryption algorithm**

that takes the ciphertext and Bob's key as input and produces the original plaintext as output. That is, if Bob receives an encrypted message $K_A(m)$, he decrypts it by computing $K_B(K_A(m)) = m$. In **symmetric key systems**, Alice's and Bob's keys are identical and are secret. In **public key systems**, a pair of keys is used. One of the keys is known to both Bob and Alice (indeed, it is known to the whole world). The other key is known only by either Bob or Alice (but not both). In the following two subsections, we consider symmetric key and public key systems in more detail.

8.2.1 Symmetric Key Cryptography

All cryptographic algorithms involve substituting one thing for another, for example, taking a piece of plaintext and then computing and substituting the appropriate ciphertext to create the encrypted message. Before studying a modern key-based cryptographic system, let us first get our feet wet by studying a very old, very simple symmetric key algorithm attributed to Julius Caesar, known as the **Caesar cipher** (a cipher is a method for encrypting data).

For English text, the Caesar cipher would work by taking each letter in the plaintext message and substituting the letter that is k letters later (allowing wraparound; that is, having the letter z followed by the letter a) in the alphabet. For example if $k = 3$, then the letter a in plaintext becomes d in ciphertext; b in plaintext becomes e in ciphertext, and so on. Here, the value of k serves as the key. As an example, the plaintext message "`bob, i love you. Alice`" becomes "`ere, l oryh brx. dolfh`" in ciphertext. While the ciphertext does indeed look like gibberish, it wouldn't take long to break the code if you knew that the Caesar cipher was being used, as there are only 25 possible key values.

An improvement on the Caesar cipher is the **monoalphabetic cipher**, which also substitutes one letter of the alphabet with another letter of the alphabet. However, rather than substituting according to a regular pattern (for example, substitution with an offset of k for all letters), any letter can be substituted for any other letter, as long as each letter has a unique substitute letter, and vice versa. The substitution rule in Figure 8.3 shows one possible rule for encoding plaintext.

The plaintext message "`bob, i love you. Alice`" becomes "`nkn, s gktc wky. Mgsbc`." Thus, as in the case of the Caesar cipher, this looks like gibberish. A monoalphabetic cipher would also appear to be better than the Caesar cipher in that there are 26! (on the order of 10^{26}) possible pairings of letters rather than 25 possible pairings. A brute-force approach of trying all 10^{26} possible pairings

Plaintext letter:	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Ciphertext letter:	m	n	b	v	c	x	z	a	s	d	f	g	h	j	k	l	p	o	i	u	y	t	r	e	w	q

Figure 8.3 ♦ A monoalphabetic cipher

would require far too much work to be a feasible way of breaking the encryption algorithm and decoding the message. However, by statistical analysis of the plaintext language, for example, knowing that the letters *e* and *t* are the most frequently occurring letters in typical English text (accounting for 13 percent and 9 percent of letter occurrences), and knowing that particular two-and three-letter occurrences of letters appear quite often together (for example, "in," "it," "the," "ion," "ing," and so forth) make it relatively easy to break this code. If the intruder has some knowledge about the possible contents of the message, then it is even easier to break the code. For example, if Trudy the intruder is Bob's wife and suspects Bob of having an affair with Alice, then she might suspect that the names "bob" and "alice" appear in the text. If Trudy knew for certain that those two names appeared in the ciphertext and had a copy of the example ciphertext message above, then she could immediately determine seven of the 26 letter pairings, requiring 10^9 fewer possibilities to be checked by a brute-force method. Indeed, if Trudy suspected Bob of having an affair, she might well expect to find some other choice words in the message as well.

When considering how easy it might be for Trudy to break Bob and Alice's encryption scheme, one can distinguish three different scenarios, depending on what information the intruder has.

- *Ciphertext-only attack.* In some cases, the intruder may have access only to the intercepted ciphertext, with no certain information about the contents of the plaintext message. We have seen how statistical analysis can help in a **ciphertext-only attack** on an encryption scheme.
- *Known-plaintext attack.* We saw above that if Trudy somehow knew for sure that "bob" and "alice" appeared in the ciphertext message, then she could have determined the (plaintext, ciphertext) pairings for the letters *a, l, i, c, e, b,* and *o*. Trudy might also have been fortunate enough to have recorded all of the ciphertext transmissions and then found Bob's own decrypted version of one of the transmissions scribbled on a piece of paper. When an intruder knows some of the (plaintext, ciphertext) pairings, we refer to this as a **known-plaintext attack** on the encryption scheme.
- *Chosen-plaintext attack.* In a **chosen-plaintext attack**, the intruder is able to choose the plaintext message and obtain its corresponding ciphertext form. For the simple encryption algorithms we've seen so far, if Trudy could get Alice to send the message, "`The quick brown fox jumps over the lazy dog`," she could completely break the encryption scheme. We'll see shortly that for more sophisticated encryption techniques, a chosen-plaintext attack does not necessarily mean that the encryption technique can be broken.

Five hundred years ago, techniques improving on monoalphabetic encryption, known as **polyalphabetic encryption**, were invented. The idea behind polyalphabetic encryption is to use multiple monoalphabetic ciphers, with a specific

Plaintext letter:	a b c d e f g h i j k l m n o p q r s t u v w x y z
$C_1(k = 5)$:	f g h i j k l m n o p q r s t u v w x y z a b c d e
$C_2(k = 19)$:	t u v w x y z a b c d e f g h i j k l m n o p q r s

Figure 8.4 ♦ A polyalphabetic cipher using two Caesar ciphers

monoalphabetic cipher to encode a letter in a specific position in the plaintext message. Thus, the same letter, appearing in different positions in the plaintext message, might be encoded differently. An example of a polyalphabetic encryption scheme is shown in Figure 8.4. It has two Caesar ciphers (with $k = 5$ and $k = 19$), shown as rows. We might choose to use these two Caesar ciphers, C_1 and C_2, in the repeating pattern C_1, C_2, C_2, C_1, C_2. That is, the first letter of plaintext is to be encoded using C_1, the second and third using C_2, the fourth using C_1, and the fifth using C_2. The pattern then repeats, with the sixth letter being encoded using C_1, the seventh with C_2, and so on. The plaintext message "`bob, i love you.`" is thus encrypted "`ghu, n etox dhz.`" Note that the first *b* in the plaintext message is encrypted using C_1, while the second *b* is encrypted using C_2. In this example, the encryption and decryption "key" is the knowledge of the two Caesar keys ($k = 5, k = 19$) and the pattern C_1, C_2, C_2, C_1, C_2.

Block Ciphers

Let us now move forward to modern times and examine how symmetric key encryption is done today. There are two broad classes of symmetric encryption techniques: **stream ciphers** and **block ciphers**. We'll briefly examine stream ciphers in Section 8.7 when we investigate security for wireless LANs. In this section, we focus on block ciphers, which are used in many secure Internet protocols, including PGP (for secure e-mail), SSL (for securing TCP connections), and IPsec (for securing the network-layer transport).

In a block cipher, the message to be encrypted is processed in blocks of *k* bits. For example, if $k = 64$, then the message is broken into 64-bit blocks, and each block is encrypted independently. To encode a block, the cipher uses a one-to-one mapping to map the *k*-bit block of cleartext to a *k*-bit block of ciphertext. Let's look at an example. Suppose that $k = 3$, so that the block cipher maps 3-bit inputs (cleartext) to 3-bit outputs (ciphertext). One possible mapping is given in Table 8.1. Notice that this is a one-to-one mapping; that is, there is a different output for each input. This block cipher breaks the message up into 3-bit blocks and encrypts each block according to the above mapping. You should verify that the message 010110001111 gets encrypted into 101000111001.

Continuing with this 3-bit block example, note that the mapping in Table 8.1 is just one mapping of many possible mappings. How many possible mappings are

input	output	input	output
000	110	100	011
001	111	101	010
010	101	110	000
011	100	111	001

Table 8.1 ♦ A specific 3-bit block cipher

there? To answer this question, observe that a mapping is nothing more than a permutation of all the possible inputs. There are 2^3 (= 8) possible inputs (listed under the input columns). These eight inputs can be permuted in 8! = 40,320 different ways. Since each of these permutations specifies a mapping, there are 40,320 possible mappings. We can view each of these mappings as a key—if Alice and Bob both know the mapping (the key), they can encrypt and decrypt the messages sent between them.

The brute-force attack for this cipher is to try to decrypt ciphertext by using all mappings. With only 40,320 mappings (when $k = 3$), this can quickly be accomplished on a desktop PC. To thwart brute-force attacks, block ciphers typically use much larger blocks, consisting of $k = 64$ bits or even larger. Note that the number of possible mappings for a general k-block cipher is $2^k!$, which is astronomical for even moderate values of k (such as $k = 64$).

Although full-table block ciphers, as just described, with moderate values of k can produce robust symmetric key encryption schemes, they are unfortunately difficult to implement. For $k = 64$ and for a given mapping, Alice and Bob would need to maintain a table with 2^{64} input values, which is an infeasible task. Moreover, if Alice and Bob were to change keys, they would have to each regenerate the table. Thus, a full-table block cipher, providing predetermined mappings between all inputs and outputs (as in the example above), is simply out of the question.

Instead, block ciphers typically use functions that simulate randomly permuted tables. An example (adapted from [Kaufman 1995]) of such a function for $k = 64$ bits is shown in Figure 8.5. The function first breaks a 64-bit block into 8 chunks, with each chunk consisting of 8 bits. Each 8-bit chunk is processed by an 8-bit to 8-bit table, which is of manageable size. For example, the first chunk is processed by the table denoted by T_1. Next, the 8 output chunks are reassembled into a 64-bit block. The positions of the 64 bits in the block are then scrambled (permuted) to produce a 64-bit output. This output is fed back to the 64-bit input, where another cycle begins. After n such cycles, the function provides a 64-bit block of ciphertext. The purpose of the rounds is to make each input bit affect most (if not all) of the final output bits. (If only one round were used, a given input bit would affect only 8 of the 64 output bits.) The key for this block cipher algorithm would be the eight permutation tables (assuming the scramble function is publicly known).

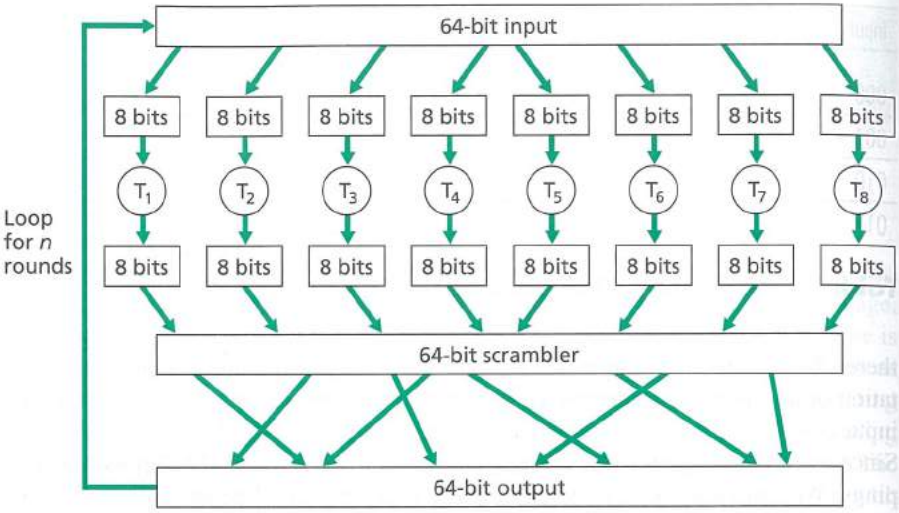


Figure 8.5 ♦ An example of a block cipher

Today there are a number of popular block ciphers, including DES (standing for Data Encryption Standard), 3DES, and AES (standing for Advanced Encryption Standard). Each of these standards uses functions, rather than predetermined tables, along the lines of Figure 8.5 (albeit more complicated and specific to each cipher). Each of these algorithms also uses a string of bits for a key. For example, DES uses 64-bit blocks with a 56-bit key. AES uses 128-bit blocks and can operate with keys that are 128, 192, and 256 bits long. An algorithm's key determines the specific "mini-table" mappings and permutations within the algorithm's internals. The brute-force attack for each of these ciphers is to cycle through all the keys, applying the decryption algorithm with each key. Observe that with a key length of n, there are 2^n possible keys. NIST [NIST 2001] estimates that a machine that could crack 56-bit DES in one second (that is, try all 2^{56} keys in one second) would take approximately 149 trillion years to crack a 128-bit AES key.

Cipher-Block Chaining

In computer networking applications, we typically need to encrypt long messages (or long streams of data). If we apply a block cipher as described by simply chopping up the message into k-bit blocks and independently encrypting each block, a subtle but important problem occurs. To see this, observe that two or more of the cleartext blocks can be identical. For example, the cleartext in two or more blocks could be "HTTP/1.1". For these identical blocks, a block cipher would, of course, produce the same ciphertext. An attacker could potentially guess the cleartext when it sees identical ciphertext blocks and may even be able to decrypt the entire message by

identifying identical ciphertext blocks and using knowledge about the underlying protocol structure [Kaufman 1995].

To address this problem, we can mix some randomness into the ciphertext so that identical plaintext blocks produce different ciphertext blocks. To explain this idea, let $m(i)$ denote the ith plaintext block, $c(i)$ denote the ith ciphertext block, and $a \oplus b$ denote the exclusive-or (XOR) of two bit strings, a and b. (Recall that the $0 \oplus 0 = 1 \oplus 1 = 0$ and $0 \oplus 1 = 1 \oplus 0 = 1$, and the XOR of two bit strings is done on a bit-by-bit basis. So, for example, $10101010 \oplus 11110000 = 01011010$.) Also, denote the block-cipher encryption algorithm with key S as K_S. The basic idea is as follows. The sender creates a random k-bit number $r(i)$ for the ith block and calculates $c(i) = K_S(m(i) \oplus r(i))$. Note that a new k-bit random number is chosen for each block. The sender then sends $c(1)$, $r(1)$, $c(2)$, $r(2)$, $c(3)$, $r(3)$, and so on. Since the receiver receives $c(i)$ and $r(i)$, it can recover each block of the plaintext by computing $m(i) = K_S(c(i)) \oplus r(i)$. It is important to note that, although $r(i)$ is sent in the clear and thus can be sniffed by Trudy, she cannot obtain the plaintext $m(i)$, since she does not know the key K_S. Also note that if two plaintext blocks $m(i)$ and $m(j)$ are the same, the corresponding ciphertext blocks $c(i)$ and $c(j)$ will be different (as long as the random numbers $r(i)$ and $r(j)$ are different, which occurs with very high probability).

As an example, consider the 3-bit block cipher in Table 8.1. Suppose the plaintext is 010010010. If Alice encrypts this directly, without including the randomness, the resulting ciphertext becomes 101101101. If Trudy sniffs this ciphertext, because each of the three cipher blocks is the same, she can correctly surmise that each of the three plaintext blocks are the same. Now suppose instead Alice generates the random blocks $r(1) = 001$, $r(2) = 111$, and $r(3) = 100$ and uses the above technique to generate the ciphertext $c(1) = 100$, $c(2) = 010$, and $c(3) = 000$. Note that the three ciphertext blocks are different even though the plaintext blocks are the same. Alice then sends $c(1)$, $r(1)$, $c(2)$, and $r(2)$. You should verify that Bob can obtain the original plaintext using the shared key K_S.

The astute reader will note that introducing randomness solves one problem but creates another: namely, Alice must transmit twice as many bits as before. Indeed, for each cipher bit, she must now also send a random bit, doubling the required bandwidth. In order to have our cake and eat it too, block ciphers typically use a technique called **Cipher Block Chaining (CBC)**. The basic idea is to send only *one random value along with the very first message, and then have the sender and receiver use the computed coded blocks in place of the subsequent random number.* Specifically, CBC operates as follows:

1. Before encrypting the message (or the stream of data), the sender generates a random k-bit string, called the **Initialization Vector (IV)**. Denote this initialization vector by $c(0)$. The sender sends the IV to the receiver *in cleartext.*
2. For the first block, the sender calculates $m(1) \oplus c(0)$, that is, calculates the exclusive-or of the first block of cleartext with the IV. It then runs the result

through the block-cipher algorithm to get the corresponding ciphertext block; that is, $c(1) = K_S(m(1) \oplus c(0))$. The sender sends the encrypted block $c(1)$ to the receiver.

3. For the ith block, the sender generates the ith ciphertext block from $c(i) = K_S(m(i) \oplus c(i-1))$.

Let's now examine some of the consequences of this approach. First, the receiver will still be able to recover the original message. Indeed, when the receiver receives $c(i)$, it decrypts it with K_S to obtain $s(i) = m(i) \oplus c(i-1)$; since the receiver also knows $c(i-1)$, it then obtains the cleartext block from $m(i) = s(i) \oplus c(i-1)$. Second, even if two cleartext blocks are identical, the corresponding ciphtertexts (almost always) will be different. Third, although the sender sends the IV in the clear, an intruder will still not be able to decrypt the ciphertext blocks, since the intruder does not know the secret key, S. Finally, the sender only sends one overhead block (the IV), thereby negligibly increasing the bandwidth usage for long messages (consisting of hundreds of blocks).

As an example, let's now determine the ciphertext for the 3-bit block cipher in Table 8.1 with plaintext 010010010 and IV $= c(0) = 001$. The sender first uses the IV to calculate $c(1) = K_S(m(1) \oplus c(0)) = 100$. The sender then calculates $c(2) = K_S(m(2) \oplus c(1)) = K_S(010 \oplus 100) = 000$, and $c(3) = K_S(m(3) \oplus c(2)) = K_S(010 \oplus 000) = 101$. The reader should verify that the receiver, knowing the IV and K_S can recover the original plaintext.

CBC has an important consequence when designing secure network protocols: we'll need to provide a mechanism within the protocol to distribute the IV from sender to receiver. We'll see how this is done for several protocols later in this chapter.

8.2.2 Public Key Encryption

For more than 2,000 years (since the time of the Caesar cipher and up to the 1970s), encrypted communication required that the two communicating parties share a common secret—the symmetric key used for encryption and decryption. One difficulty with this approach is that the two parties must somehow agree on the shared key; but to do so requires (presumably *secure*) communication! Perhaps the parties could first meet and agree on the key in person (for example, two of Caesar's centurions might meet at the Roman baths) and thereafter communicate with encryption. In a networked world, however, communicating parties may never meet and may never converse except over the network. Is it possible for two parties to communicate with encryption without having a shared secret key that is known in advance? In 1976, Diffie and Hellman [Diffie 1976] demonstrated an algorithm (known now as Diffie-Hellman Key Exchange) to do just that—a radically different and marvelously elegant approach toward secure communication that has led to the development of today's public key cryptography systems. We'll see shortly that public key cryptography systems also have several wonderful properties that make them useful not only

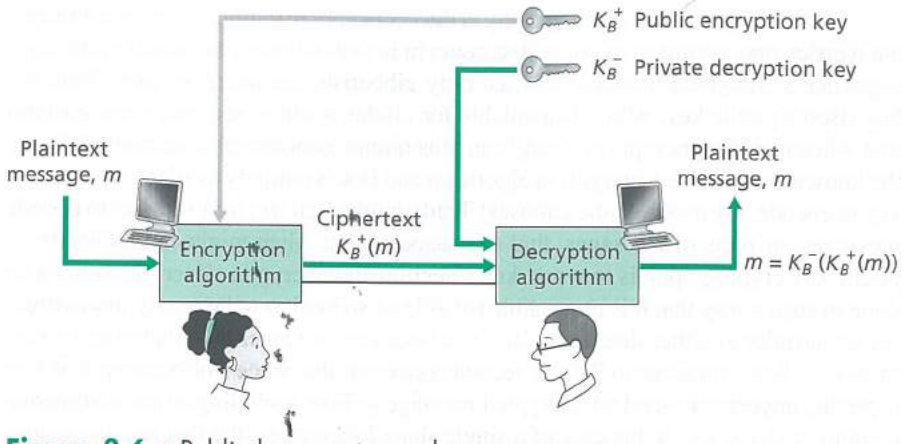


Figure 8.6 ♦ Public key cryptography

for encryption, but for authentication and digital signatures as well. Interestingly, it has recently come to light that ideas similar to those in [Diffie 1976] and [RSA 1978] had been independently developed in the early 1970s in a series of secret reports by researchers at the Communications-Electronics Security Group in the United Kingdom [Ellis 1987]. As is often the case, great ideas can spring up independently in many places; fortunately, public key advances took place not only in private, but also in the public view, as well.

The use of public key cryptography is conceptually quite simple. Suppose Alice wants to communicate with Bob. As shown in Figure 8.6, rather than Bob and Alice sharing a single secret key (as in the case of symmetric key systems), Bob (the recipient of Alice's messages) instead has two keys—a **public key** that is available to *everyone* in the world (including Trudy the intruder) and a **private key** that is known only to Bob. We will use the notation K_B^+ and K_B^- to refer to Bob's public and private keys, respectively. In order to communicate with Bob, Alice first fetches Bob's public key. Alice then encrypts her message, m, to Bob using Bob's public key and a known (for example, standardized) encryption algorithm; that is, Alice computes $K_B^+(m)$. Bob receives Alice's encrypted message and uses his private key and a known (for example, standardized) decryption algorithm to decrypt Alice's encrypted message. That is, Bob computes $K_B^-(K_B^+(m))$. We will see below that there are encryption/decryption algorithms and techniques for choosing public and private keys such that $K_B^-(K_B^+(m)) = m$; that is, applying Bob's public key, K_B^+, to a message, m (to get $K_B^+(m)$), and then applying Bob's private key, K_B^-, to the encrypted version of m (that is, computing $K_B^-(K_B^+(m))$) gives back m. This is a remarkable result! In this manner, Alice can use Bob's publicly available key to send a secret message to Bob without either of them having to distribute any secret keys! We will see shortly that we can interchange the public key and private key encryption and get the same remarkable result—that is, $K_B^-(_B{}^+(m)) = K_B^+(K_B^-(m)) = m$.

The use of public key cryptography is thus conceptually simple. But two immediate worries may spring to mind. A first concern is that although an intruder intercepting Alice's encrypted message will see only gibberish, the intruder knows both the key (Bob's public key, which is available for all the world to see) and the algorithm that Alice used for encryption. Trudy can thus mount a chosen-plaintext attack, using the known standardized encryption algorithm and Bob's publicly available encryption key to encode any message she chooses! Trudy might well try, for example, to encode messages, or parts of messages, that she suspects that Alice might send. Clearly, if public key cryptography is to work, key selection and encryption/decryption must be done in such a way that it is impossible (or at least so hard as to be nearly impossible) for an intruder to either determine Bob's private key or somehow otherwise decrypt or guess Alice's message to Bob. A second concern is that since Bob's encryption key is public, anyone can send an encrypted message to Bob, including Alice or someone *claiming* to be Alice. In the case of a single shared secret key, the fact that the sender knows the secret key implicitly identifies the sender to the receiver. In the case of public key cryptography, however, this is no longer the case since anyone can send an encrypted message to Bob using Bob's publicly available key. A digital signature, a topic we will study in Section 8.3, is needed to bind a sender to a message.

RSA

While there may be many algorithms that address these concerns, the **RSA algorithm** (named after its founders, Ron Rivest, Adi Shamir, and Leonard Adleman) has become almost synonymous with public key cryptography. Let's first see how RSA works and then examine why it works.

RSA makes extensive use of arithmetic operations using modulo-n arithmetic. So let's briefly review modular arithmetic. Recall that $x \bmod n$ simply means the remainder of x when divided by n; so, for example, $19 \bmod 5 = 4$. In modular arithmetic, one performs the usual operations of addition, multiplication, and exponentiation. However, the result of each operation is replaced by the integer remainder that is left when the result is divided by n. Adding and multiplying with modular arithmetic is facilitated with the following handy facts:

$$[(a \bmod n) + (b \bmod n)] \bmod n = (a + b) \bmod n$$

$$[(a \bmod n) - (b \bmod n)] \bmod n = (a - b) \bmod n$$

$$[(a \bmod n) \cdot (b \bmod n)] \bmod n = (a \cdot b) \bmod n$$

It follows from the third fact that $(a \bmod n)^d \bmod n = a^d \bmod n$, which is an identity that we will soon find very useful.

Now suppose that Alice wants to send to Bob an RSA-encrypted message, as shown in Figure 8.6. In our discussion of RSA, let's always keep in mind that a message is nothing but a bit pattern, and every bit pattern can be uniquely represented by

an integer number (along with the length of the bit pattern). For example, suppose a message is the bit pattern 1001; this message can be represented by the decimal integer 9. Thus, when encrypting a message with RSA, it is equivalent to encrypting the unique integer number that represents the message.

There are two interrelated components of RSA:

- The choice of the public key and the private key
- The encryption and decryption algorithm

To generate the public and private RSA keys, Bob performs the following steps:

1. Choose two large prime numbers, p and q. How large should p and q be? The larger the values, the more difficult it is to break RSA, but the longer it takes to perform the encoding and decoding. RSA Laboratories recommends that the product of p and q be on the order of 1,024 bits. For a discussion of how to find large prime numbers, see [Caldwell 2012].
2. Compute $n = pq$ and $z = (p - 1)(q - 1)$.
3. Choose a number, e, less than n, that has no common factors (other than 1) with z. (In this case, e and z are said to be relatively prime.) The letter e is used since this value will be used in encryption.
4. Find a number, d, such that $ed - 1$ is exactly divisible (that is, with no remainder) by z. The letter d is used because this value will be used in decryption. Put another way, given e, we choose d such that

$$ed \bmod z = 1$$

5. The public key that Bob makes available to the world, K_B^+, is the pair of numbers (n, e); his private key, K_B^-, is the pair of numbers (n, d).

The encryption by Alice and the decryption by Bob are done as follows:

- Suppose Alice wants to send Bob a bit pattern represented by the integer number m (with $m < n$). To encode, Alice performs the exponentiation m^e, and then computes the integer remainder when m^e is divided by n. In other words, the encrypted value, c, of Alice's plaintext message, m, is

$$c = m^e \bmod n$$

  The bit pattern corresponding to this ciphertext c is sent to Bob.
- To decrypt the received ciphertext message, c, Bob computes

$$m = c^d \bmod n$$

  which requires the use of his private key (n, d).

Plaintext Letter	m: numeric representation	m^e	Ciphertext $c = m^e \bmod n$
l	12	248832	17
o	15	759375	15
v	22	5153632	22
e	5	3125	10

Table 8.2 ♦ Alice's RSA encryption, $e = 5$, $n = 35$

As a simple example of RSA, suppose Bob chooses $p = 5$ and $q = 7$. (Admittedly, these values are far too small to be secure.) Then $n = 35$ and $z = 24$. Bob chooses $e = 5$, since 5 and 24 have no common factors. Finally, Bob chooses $d = 29$, since $5 \cdot 29 - 1$ (that is, $ed - 1$) is exactly divisible by 24. Bob makes the two values, $n = 35$ and $e = 5$, public and keeps the value $d = 29$ secret. Observing these two public values, suppose Alice now wants to send the letters l, o, v, and e to Bob. Interpreting each letter as a number between 1 and 26 (with a being 1, and z being 26), Alice and Bob perform the encryption and decryption shown in Tables 8.2 and 8.3, respectively. Note that in this example, we consider each of the four letters as a distinct message. A more realistic example would be to convert the four letters into their 8-bit ASCII representations and then encrypt the integer corresponding to the resulting 32-bit bit pattern. (Such a realistic example generates numbers that are much too long to print in a textbook!)

Given that the "toy" example in Tables 8.2 and 8.3 has already produced some extremely large numbers, and given that we saw earlier that p and q should each be several hundred bits long, several practical issues regarding RSA come to mind. How does one choose large prime numbers? How does one then choose e and d? How does one perform exponentiation with large numbers? A discussion of these important issues is beyond the scope of this book; see [Kaufman 1995] and the references therein for details.

Ciphertext c	c^d	$m = c^d \bmod n$	Plaintext Letter
17	48196857210675091509150914118252230716 97	12	l
15	12783403940394885893911123275756835937 5	15	o
22	85164331908653770195619449972110603059 2	22	v
10	10000000000000000000000000000000	5	e

Table 8.3 ♦ Bob's RSA decryption, $d = 29$, $n = 35$

Session Keys

We note here that the exponentiation required by RSA is a rather time-consuming process. By contrast, DES is at least 100 times faster in software and between 1,000 and 10,000 times faster in hardware [RSA Fast 2012]. As a result, RSA is often used in practice in combination with symmetric key cryptography. For example, if Alice wants to send Bob a large amount of encrypted data, she could do the following. First Alice chooses a key that will be used to encode the data itself; this key is referred to as a **session key**, and is denoted by K_S. Alice must inform Bob of the session key, since this is the shared symmetric key they will use with a symmetric key cipher (e.g., with DES or AES). Alice encrypts the session key using Bob's public key, that is, computes $c = (K_S)^e \bmod n$. Bob receives the RSA-encrypted session key, c, and decrypts it to obtain the session key, K_S. Bob now knows the session key that Alice will use for her encrypted data transfer.

Why Does RSA Work?

RSA encryption/decryption appears rather magical. Why should it be that by applying the encryption algorithm and then the decryption algorithm, one recovers the original message? In order to understand why RSA works, again denote $n = pq$, where p and q are the large prime numbers used in the RSA algorithm.

Recall that, under RSA encryption, a message (uniquely represented by an integer), n, is exponentiated to the power e using modulo-n arithmetic, that is,

$$c = m^e \bmod n$$

Decryption is performed by raising this value to the power d, again using modulo-n arithmetic. The result of an encryption step followed by a decryption step is thus $m^e \bmod n)^d \bmod n$. Let's now see what we can say about this quantity. As mentioned earlier, one important property of modulo arithmetic is $(a \bmod n)^d \bmod n = a^d \bmod n$ for any values a, n, and d. Thus, using $a = m^e$ in this property, we have

$$(m^e \bmod n)^d \bmod n = m^{ed} \bmod n$$

It therefore remains to show that $m^{ed} \bmod n = m$. Although we're trying to remove some of the magic about why RSA works, to establish this, we'll need to use a rather magical result from number theory here. Specifically, we'll need the result that says if p and q are prime, $n = pq$, and $z = (p - 1)(q - 1)$, then $x^y \bmod n$ is the same as $x^{(y \bmod z)} \bmod n$ [Kaufman 1995]. Applying this result with $x = m$ and $y = ed$ we have

$$m^{ed} \bmod n = m^{(ed \bmod z)} \bmod n$$

But remember that we have chosen e and d such that $ed \bmod z = 1$. This gives us

$$m^{ed} \bmod n = m^1 \bmod n = m$$

which is exactly the result we are looking for! By first exponentiating to the power of e (that is, encrypting) and then exponentiating to the power of d (that is, decrypting), we obtain the original value, m. Even *more* wonderful is the fact that if we first exponentiate to the power of d and then exponentiate to the power of e—that is, we reverse the order of encryption and decryption, performing the decryption operation first and then applying the encryption operation—we also obtain the original value, m. This wonderful result follows immediately from the modular arithmetic:

$$(m^d \bmod n)^e \bmod n = m^{de} \bmod n = m^{ed} \bmod n = (m^e \bmod n)^d \bmod n$$

The security of RSA relies on the fact that there are no known algorithms for quickly factoring a number, in this case the public value n, into the primes p and q. If one knew p and q, then given the public value e, one could easily compute the secret key, d. On the other hand, it is not known whether or not there *exist* fast algorithms for factoring a number, and in this sense, the security of RSA is not guaranteed.

Another popular public-key encryption algorithm is the Diffie-Hellman algorithm, which we will briefly explore in the homework problems. Diffie-Hellman is not as versatile as RSA in that it cannot be used to encrypt messages of arbitrary length; it can be used, however, to establish a symmetric session key, which is in turn used to encrypt messages.

8.3 Message Integrity and Digital Signatures

In the previous section we saw how encryption can be used to provide confidentiality to two communicating entities. In this section we turn to the equally important cryptography topic of providing **message integrity** (also known as message authentication). Along with message integrity, we will discuss two related topics in this section: digital signatures and end-point authentication.

We define the message integrity problem using, once again, Alice and Bob. Suppose Bob receives a message (which may be encrypted or may be in plaintext) and he believes this message was sent by Alice. To authenticate this message, Bob needs to verify:

1. The message indeed originated from Alice.
2. The message was not tampered with on its way to Bob.

We'll see in Sections 8.4 through 8.7 that this problem of message integrity is a critical concern in just about all secure networking protocols.

As a specific example, consider a computer network using a link-state routing algorithm (such as OSPF) for determining routes between each pair of routers in the

network (see Chapter 5). In a link-state algorithm, each router needs to broadcast a link-state message to all other routers in the network. A router's link-state message includes a list of its directly connected neighbors and the direct costs to these neighbors. Once a router receives link-state messages from all of the other routers, it can create a complete map of the network, run its least-cost routing algorithm, and configure its forwarding table. One relatively easy attack on the routing algorithm is for Trudy to distribute bogus link-state messages with incorrect link-state information. Thus the need for message integrity—when router B receives a link-state message from router A, router B should verify that router A actually created the message and, further, that no one tampered with the message in transit.

In this section, we describe a popular message integrity technique that is used by many secure networking protocols. But before doing so, we need to cover another important topic in cryptography—cryptographic hash functions.

8.3.1 Cryptographic Hash Functions

As shown in Figure 8.7, a hash function takes an input, m, and computes a fixed-size string $H(m)$ known as a hash. The Internet checksum (Chapter 3) and CRCs (Chapter 6) meet this definition. A **cryptographic hash function** is required to have the following additional property:

- It is computationally infeasible to find any two different messages x and y such that $H(x) = H(y)$.

Informally, this property means that it is computationally infeasible for an intruder to substitute one message for another message that is protected by the hash

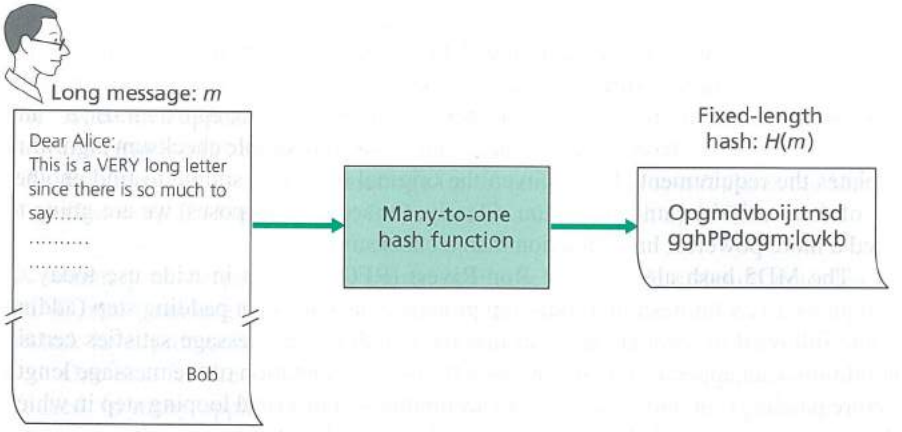


Figure 8.7 ♦ Hash functions

Message	ASCII Representation				
I O U 1	49	4F	55	31	
0 0 . 9	30	30	2E	39	
9 B O B	39	42	4F	42	
	B2	C1	D2	AC	Checksum

Message	ASCII Representation				
I O U 9	49	4F	55	39	
0 0 . 1	30	30	2E	31	
9 B O B	39	42	4F	42	
	B2	C1	D2	AC	Checksum

Figure 8.8 ♦ Initial message and fraudulent message have the same checksum!

function. That is, if $(m, H(m))$ are the message and the hash of the message created by the sender, then an intruder cannot forge the contents of another message, y, that has the same hash value as the original message.

Let's convince ourselves that a simple checksum, such as the Internet checksum, would make a poor cryptographic hash function. Rather than performing 1s complement arithmetic (as in the Internet checksum), let us compute a checksum by treating each character as a byte and adding the bytes together using 4-byte chunks at a time. Suppose Bob owes Alice $100.99 and sends an IOU to Alice consisting of the text string "`IOU100.99BOB`." The ASCII representation (in hexadecimal notation) for these letters is `49,4F,55,31,30,30,2E,39,39,42,4F,42`.

Figure 8.8 (top) shows that the 4-byte checksum for this message is B2 C1 D2 AC. A slightly different message (and a much more costly one for Bob) is shown in the bottom half of Figure 8.8. The messages "`IOU100.99BOB`" and "`IOU900.19BOB`" have the *same* checksum. Thus, this simple checksum algorithm violates the requirement above. Given the original data, it is simple to find another set of data with the same checksum. Clearly, for security purposes, we are going to need a more powerful hash function than a checksum.

The MD5 hash algorithm of Ron Rivest [RFC 1321] is in wide use today. It computes a 128-bit hash in a four-step process consisting of a padding step (adding a one followed by enough zeros so that the length of the message satisfies certain conditions), an append step (appending a 64-bit representation of the message length before padding), an initialization of an accumulator, and a final looping step in which the message's 16-word blocks are processed (mangled) in four rounds. For a description of MD5 (including a C source code implementation) see [RFC 1321].

The second major hash algorithm in use today is the Secure Hash Algorithm (SHA-1) [FIPS 1995]. This algorithm is based on principles similar to those used in the design of MD4 [RFC 1320], the predecessor to MD5. SHA-1, a US federal standard, is required for use whenever a cryptographic hash algorithm is needed for federal applications. It produces a 160-bit message digest. The longer output length makes SHA-1 more secure.

8.3.2 Message Authentication Code

Let's now return to the problem of message integrity. Now that we understand hash functions, let's take a first stab at how we might perform message integrity:

1. Alice creates message m and calculates the hash $H(m)$ (for example with SHA-1).
2. Alice then appends $H(m)$ to the message m, creating an extended message $(m, H(m))$, and sends the extended message to Bob.
3. Bob receives an extended message (m, h) and calculates $H(m)$. If $H(m) = h$, Bob concludes that everything is fine.

This approach is obviously flawed. Trudy can create a bogus message m' in which she says she is Alice, calculate $H(m')$, and send Bob $(m', H(m'))$. When Bob receives the message, everything checks out in step 3, so Bob doesn't suspect any funny business.

To perform message integrity, in addition to using cryptographic hash functions, Alice and Bob will need a shared secret s. This shared secret, which is nothing more than a string of bits, is called the **authentication key**. Using this shared secret, message integrity can be performed as follows:

1. Alice creates message m, concatenates s with m to create $m + s$, and calculates the hash $H(m + s)$ (for example with SHA-1). $H(m + s)$ is called the **message authentication code (MAC)**.
2. Alice then appends the MAC to the message m, creating an extended message $(m, H(m + s))$, and sends the extended message to Bob.
3. Bob receives an extended message (m, h) and knowing s, calculates the MAC $H(m + s)$. If $H(m + s) = h$, Bob concludes that everything is fine.

A summary of the procedure is shown in Figure 8.9. Readers should note that the MAC here (standing for "message authentication code") is not the same MAC used in link-layer protocols (standing for "medium access control")!

One nice feature of a MAC is that it does not require an encryption algorithm. Indeed, in many applications, including the link-state routing algorithm described earlier, communicating entities are only concerned with message integrity and are not concerned with message confidentiality. Using a MAC, the entities can authenticate

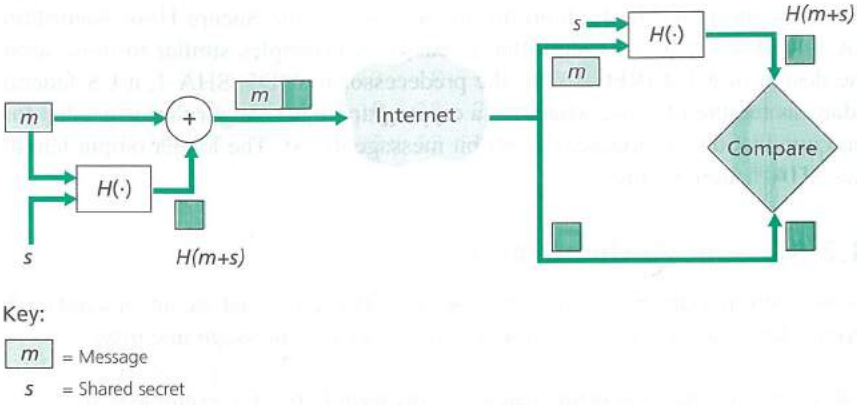


Figure 8.9 ♦ Message authentication code (MAC)

the messages they send to each other without having to integrate complex encryption algorithms into the integrity process.

As you might expect, a number of different standards for MACs have been proposed over the years. The most popular standard today is **HMAC**, which can be used either with MD5 or SHA-1. HMAC actually runs data and the authentication key through the hash function twice [Kaufman 1995; RFC 2104].

There still remains an important issue. How do we distribute the shared authentication key to the communicating entities? For example, in the link-state routing algorithm, we would somehow need to distribute the secret authentication key to each of the routers in the autonomous system. (Note that the routers can all use the same authentication key.) A network administrator could actually accomplish this by physically visiting each of the routers. Or, if the network administrator is a lazy guy, and if each router has its own public key, the network administrator could distribute the authentication key to any one of the routers by encrypting it with the router's public key and then sending the encrypted key over the network to the router.

8.3.3 Digital Signatures

Think of the number of the times you've signed your name to a piece of paper during the last week. You sign checks, credit card receipts, legal documents, and letters. Your signature attests to the fact that you (as opposed to someone else) have acknowledged and/or agreed with the document's contents. In a digital world, one often wants to indicate the owner or creator of a document, or to signify one's agreement with a document's content. A **digital signature** is a cryptographic technique for achieving these goals in a digital world.

Just as with handwritten signatures, digital signing should be done in a way that is verifiable and nonforgeable. That is, it must be possible to prove that a document

signed by an individual was indeed signed by that individual (the signature must be verifiable) and that *only* that individual could have signed the document (the signature cannot be forged).

Let's now consider how we might design a digital signature scheme. Observe that when Bob signs a message, Bob must put something on the message that is unique to him. Bob could consider attaching a MAC for the signature, where the MAC is created by appending his key (unique to him) to the message, and then taking the hash. But for Alice to verify the signature, she must also have a copy of the key, in which case the key would not be unique to Bob. Thus, MACs are not going to get the job done here.

Recall that with public-key cryptography, Bob has both a public and private key, with both of these keys being unique to Bob. Thus, public-key cryptography is an excellent candidate for providing digital signatures. Let us now examine how it is done.

Suppose that Bob wants to digitally sign a document, m. We can think of the document as a file or a message that Bob is going to sign and send. As shown in Figure 8.10, to sign this document, Bob simply uses his private key, K_B^-, to compute $K_B^-(m)$. At first, it might seem odd that Bob is using his private key (which, as we saw in Section 8.2, was used to decrypt a message that had been encrypted with his public key) to sign a document. But recall that encryption and decryption are nothing more than mathematical operations (exponentiation to the power of e or d in RSA; see Section 8.2) and recall that Bob's goal is not to scramble or obscure the contents of the document, but rather to sign the document in a manner that is verifiable and nonforgeable. Bob's digital signature of the document is $K_B^-(m)$.

Does the digital signature $K_B^-(m)$ meet our requirements of being verifiable and nonforgeable? Suppose Alice has m and $K_B^-(m)$. She wants to prove in court (being

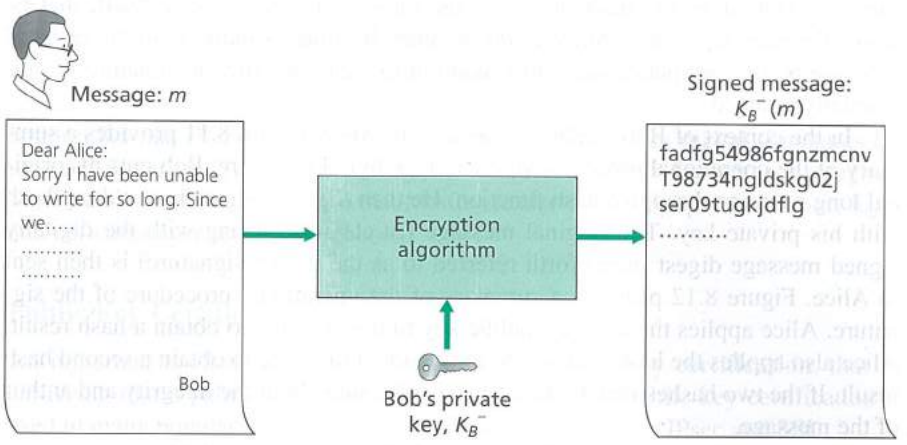


Figure 8.10 ♦ Creating a digital signature for a document

litigious) that Bob had indeed signed the document and was the only person who could have possibly signed the document. Alice takes Bob's public key, K_B^+, and applies it to the digital signature, $K_B^-(m)$, associated with the document, m. That is, she computes $K_B^+(K_B^-(m))$, and voilà, with a dramatic flurry, she produces m, which exactly matches the original document! Alice then argues that only Bob could have signed the document, for the following reasons:

- Whoever signed the message must have used the private key, K_B^-, in computing the signature $K_B^-(m)$, such that $K_B^+(K_B^-(m)) = m$.
- The only person who could have known the private key, K_B^-, is Bob. Recall from our discussion of RSA in Section 8.2 that knowing the public key, K_B^+, is of no help in learning the private key, K_B^-. Therefore, the only person who could know K_B^- is the person who generated the pair of keys, (K_B^+, K_B^-), in the first place, Bob. (Note that this assumes, though, that Bob has not given K_B^- to anyone, nor has anyone stolen K_B^- from Bob.)

It is also important to note that if the original document, m, is ever modified to some alternate form, m', the signature that Bob created for m will not be valid for m', since $K_B^+(K_B^-(m))$ does not equal m'. Thus we see that digital signatures also provide message integrity, allowing the receiver to verify that the message was unaltered as well as the source of the message.

One concern with signing data by encryption is that encryption and decryption are computationally expensive. Given the overheads of encryption and decryption, signing data via complete encryption/decryption can be overkill. A more efficient approach is to introduce hash functions into the digital signature. Recall from Section 8.3.2 that a hash algorithm takes a message, m, of arbitrary length and computes a fixed-length "fingerprint" of the message, denoted by $H(m)$. Using a hash function, Bob signs the hash of a message rather than the message itself, that is, Bob calculates $K_B^-(H(m))$. Since $H(m)$ is generally much smaller than the original message m, the computational effort required to create the digital signature is substantially reduced.

In the context of Bob sending a message to Alice, Figure 8.11 provides a summary of the operational procedure of creating a digital signature. Bob puts his original long message through a hash function. He then digitally signs the resulting hash with his private key. The original message (in cleartext) along with the digitally signed message digest (henceforth referred to as the digital signature) is then sent to Alice. Figure 8.12 provides a summary of the operational procedure of the signature. Alice applies the sender's public key to the message to obtain a hash result. Alice also applies the hash function to the cleartext message to obtain a second hash result. If the two hashes match, then Alice can be sure about the integrity and author of the message.

Before moving on, let's briefly compare digital signatures with MACs, since they have parallels, but also have important subtle differences. Both digital signatures and

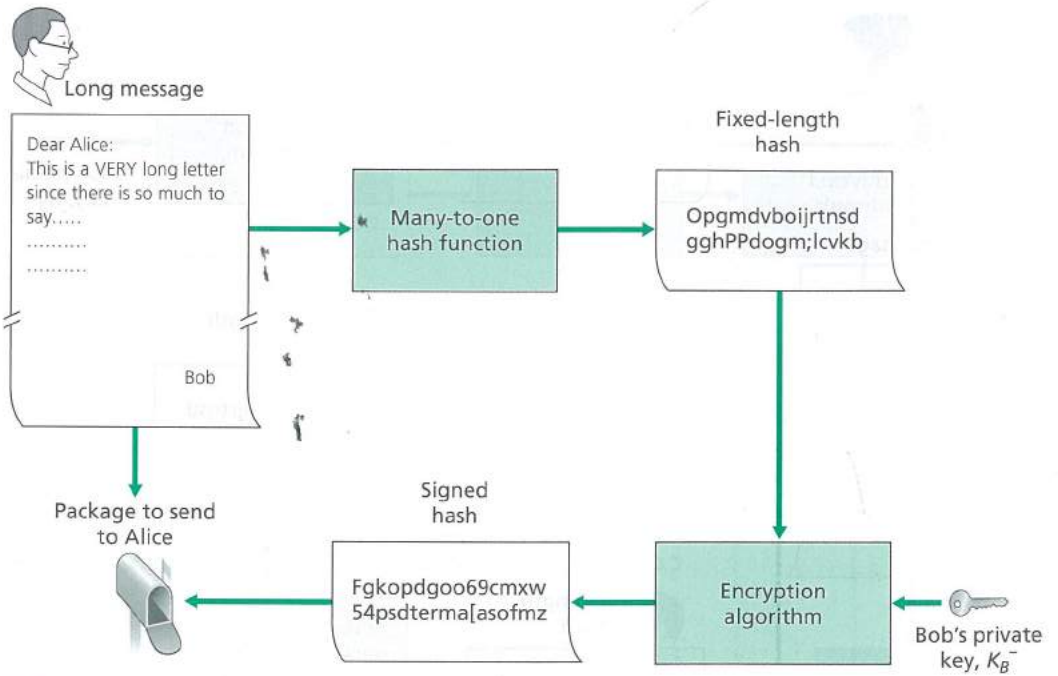


Figure 8.11 ♦ Sending a digitally signed message

MACs start with a message (or a document). To create a MAC out of the message, we append an authentication key to the message, and then take the hash of the result. Note that neither public key nor symmetric key encryption is involved in creating the MAC. To create a digital signature, we first take the hash of the message and then encrypt the message with our private key (using public key cryptography). Thus, a digital signature is a "heavier" technique, since it requires an underlying Public Key Infrastructure (PKI) with certification authorities as described below. We'll see in Section 8.4 that PGP—a popular secure e-mail system—uses digital signatures for message integrity. We've seen already that OSPF uses MACs for message integrity. We'll see in Sections 8.5 and 8.6 that MACs are also used for popular transport-layer and network-layer security protocols.

Public Key Certification

An important application of digital signatures is **public key certification**, that is, certifying that a public key belongs to a specific entity. Public key certification is used in many popular secure networking protocols, including IPsec and SSL.

To gain insight into this problem, let's consider an Internet-commerce version of the classic "pizza prank." Alice is in the pizza delivery business and accepts orders

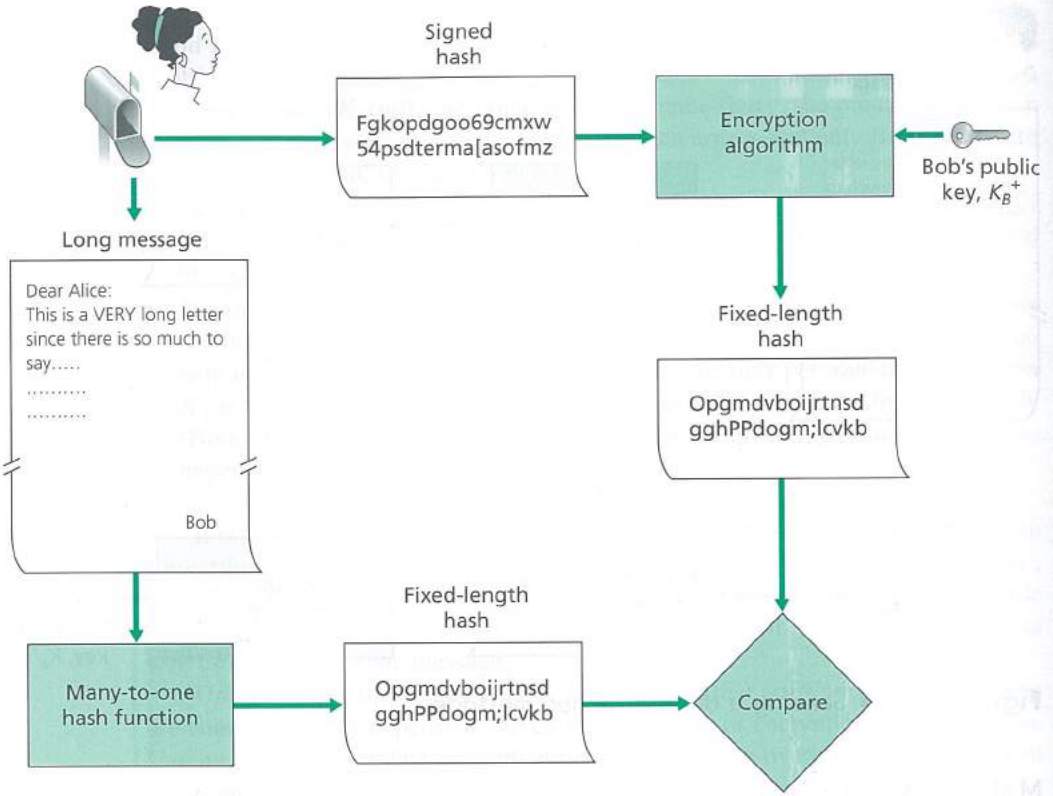


Figure 8.12 ♦ Verifying a signed message

over the Internet. Bob, a pizza lover, sends Alice a plaintext message that includes his home address and the type of pizza he wants. In this message, Bob also includes a digital signature (that is, a signed hash of the original plaintext message) to prove to Alice that he is the true source of the message. To verify the signature, Alice obtains Bob's public key (perhaps from a public key server or from the e-mail message) and checks the digital signature. In this manner she makes sure that Bob, rather than some adolescent prankster, placed the order.

This all sounds fine until clever Trudy comes along. As shown in Figure 8.13, Trudy is indulging in a prank. She sends a message to Alice in which she says she is Bob, gives Bob's home address, and orders a pizza. In this message she also includes her (Trudy's) public key, although Alice naturally assumes it is Bob's public key. Trudy also attaches a digital signature, which was created with her own (Trudy's) private key. After receiving the message, Alice applies Trudy's public key (thinking that it is Bob's) to the digital signature and concludes that the plaintext message was

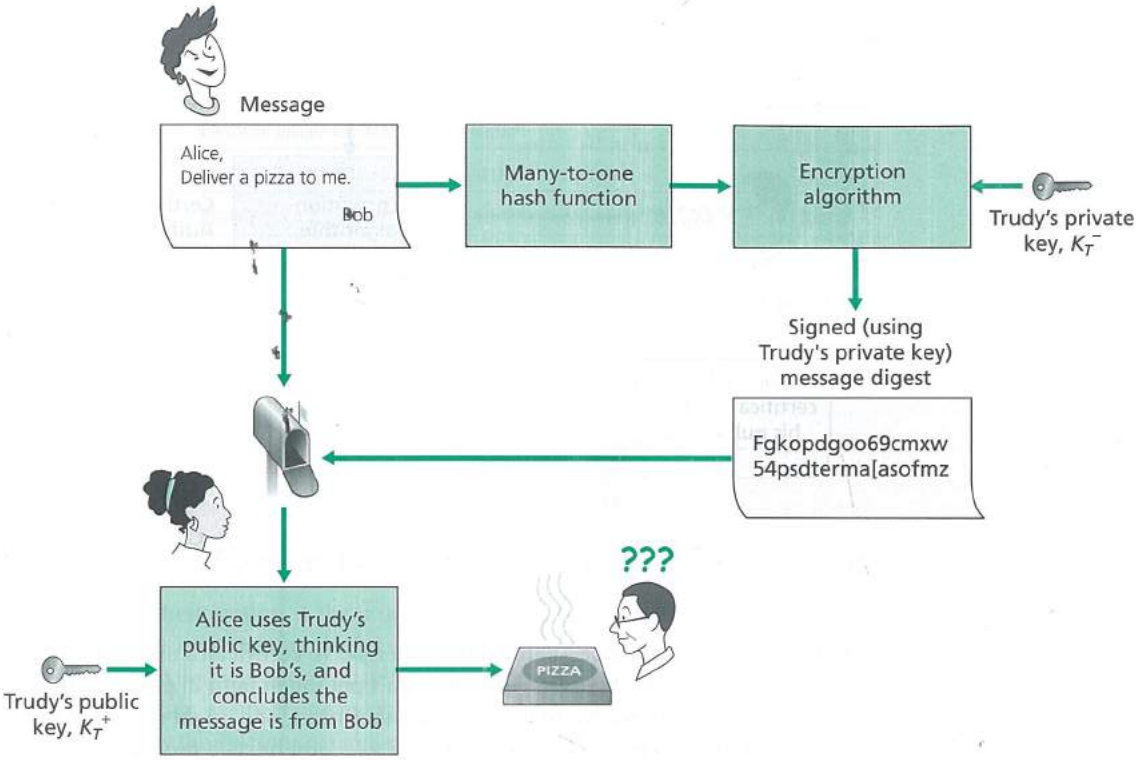


Figure 8.13 ♦ Trudy masquerades as Bob using public key cryptography

indeed created by Bob. Bob will be very surprised when the delivery person brings a pizza with pepperoni and anchovies to his home!

We see from this example that for public key cryptography to be useful, you need to be able to verify that you have the actual public key of the entity (person, router, browser, and so on) with whom you want to communicate. For example, when Alice wants to communicate with Bob using public key cryptography, she needs to verify that the public key that is supposed to be Bob's is indeed Bob's.

Binding a public key to a particular entity is typically done by a **Certification Authority (CA)**, whose job is to validate identities and issue certificates. A CA has the following roles:

1. A CA verifies that an entity (a person, a router, and so on) is who it says it is. There are no mandated procedures for how certification is done. When dealing with a CA, one must trust the CA to have performed a suitably rigorous identity verification. For example, if Trudy were able to walk into the Fly-by-Night

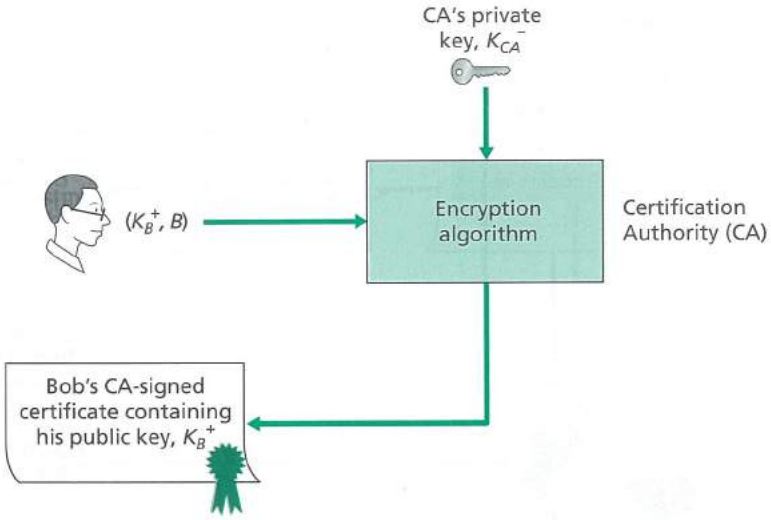


Figure 8.14 ♦ Bob has his public key certified by the CA

CA and simply announce "I am Alice" and receive certificates associated with the identity of Alice, then one shouldn't put much faith in public keys certified by the Fly-by-Night CA. On the other hand, one might (or might not!) be more willing to trust a CA that is part of a federal or state program. You can trust the identity associated with a public key only to the extent to which you can trust a CA and its identity verification techniques. What a tangled web of trust we spin!

2. Once the CA verifies the identity of the entity, the CA creates a **certificate** that binds the public key of the entity to the identity. The certificate contains the public key and globally unique identifying information about the owner of the public key (for example, a human name or an IP address). The certificate is digitally signed by the CA. These steps are shown in Figure 8.14.

Let us now see how certificates can be used to combat pizza-ordering pranksters, like Trudy, and other undesirables. When Bob places his order he also sends his CA-signed certificate. Alice uses the CA's public key to check the validity of Bob's certificate and extract Bob's public key.

Both the International Telecommunication Union (ITU) and the IETF have developed standards for CAs. ITU X.509 [ITU 2005a] specifies an authentication service as well as a specific syntax for certificates. [RFC 1422] describes CA-based key management for use with secure Internet e-mail. It is compatible with X.509 but goes beyond X.509 by establishing procedures and conventions for a key management architecture. Table 8.4 describes some of the important fields in a certificate.

Field Name	Description
Version	Version number of X.509 specification
Serial number	CA-issued unique identifier for a certificate
Signature	Specifies the algorithm used by CA to sign this certificate
Issuer name	Identity of CA issuing this certificate, in distinguished name (DN) [RFC 4514] format
Validity period	Start and end of period of validity for certificate
Subject name	Identity of entity whose public key is associated with this certificate, in DN format
Subject public key	The subject's public key as well indication of the public key algorithm (and algorithm parameters) to be used with this key

Table 8.4 ♦ Selected fields in an X.509 and RFC 1422 public key

8.4 End-Point Authentication

End-point authentication is the process of one entity proving its identity to another entity over a computer network, for example, a user proving its identity to an e-mail server. As humans, we authenticate each other in many ways: We recognize each other's faces when we meet, we recognize each other's voices on the telephone, we are authenticated by the customs official who checks us against the picture on our passport.

In this section, we consider how one party can authenticate another party when the two are communicating over a network. We focus here on authenticating a "live" party, at the point in time when communication is actually occurring. A concrete example is a user authenticating him or herself to an e-mail server. This is a subtly different problem from proving that a message received at some point in the past did indeed come from that claimed sender, as studied in Section 8.3.

When performing authentication over the network, the communicating parties cannot rely on biometric information, such as a visual appearance or a voiceprint. Indeed, we will see in our later case studies that it is often network elements such as routers and client/server processes that must authenticate each other. Here, authentication must be done solely on the basis of messages and data exchanged as part of an **authentication protocol**. Typically, an authentication protocol would run *before* the two communicating parties run some other protocol (for example, a reliable data transfer protocol, a routing information exchange protocol, or an e-mail protocol). The authentication protocol first establishes the identities of the parties to each other's satisfaction; only after authentication do the parties get down to the work at hand.

As in the case of our development of a reliable data transfer (rdt) protocol in Chapter 3, we will find it instructive here to develop various versions of an authentication protocol, which we will call **ap** (authentication protocol), and poke holes in each version

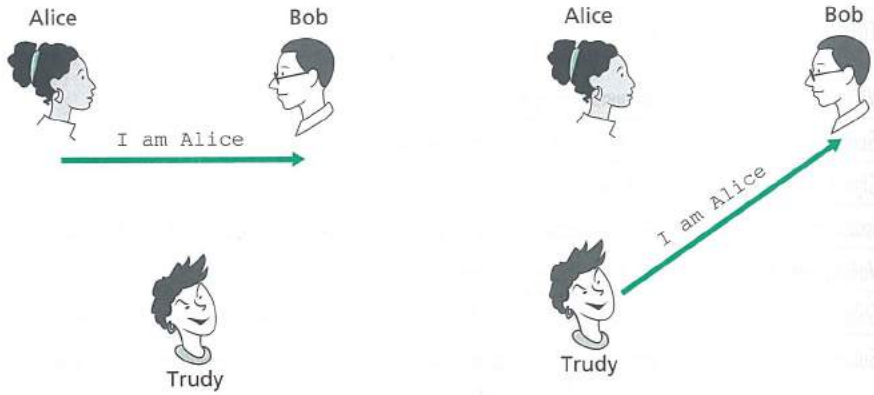


Figure 8.15 ♦ Protocol *ap1.0* and a failure scenario

as we proceed. (If you enjoy this stepwise evolution of a design, you might also enjoy [Bryant 1988], which recounts a fictitious narrative between designers of an open-network authentication system, and their discovery of the many subtle issues involved.)

Let's assume that Alice needs to authenticate herself to Bob.

8.4.1 Authentication Protocol *ap1.0*

Perhaps the simplest authentication protocol we can imagine is one where Alice simply sends a message to Bob saying she is Alice. This protocol is shown in Figure 8.15. The flaw here is obvious—there is no way for Bob actually to know that the person sending the message "I am Alice" is indeed Alice. For example, Trudy (the intruder) could just as well send such a message.

8.4.2 Authentication Protocol *ap2.0*

If Alice has a well-known network address (e.g., an IP address) from which she always communicates, Bob could attempt to authenticate Alice by verifying that the source address on the IP datagram carrying the authentication message matches Alice's well-known address. In this case, Alice would be authenticated. This might stop a very network-naive intruder from impersonating Alice, but it wouldn't stop the determined student studying this book, or many others!

From our study of the network and data link layers, we know that it is not that hard (for example, if one had access to the operating system code and could build one's own operating system kernel, as is the case with Linux and several other freely available operating systems) to create an IP datagram, put whatever IP source address we want (for example, Alice's well-known IP address) into the IP datagram, and send the datagram over the link-layer protocol to the first-hop router. From then

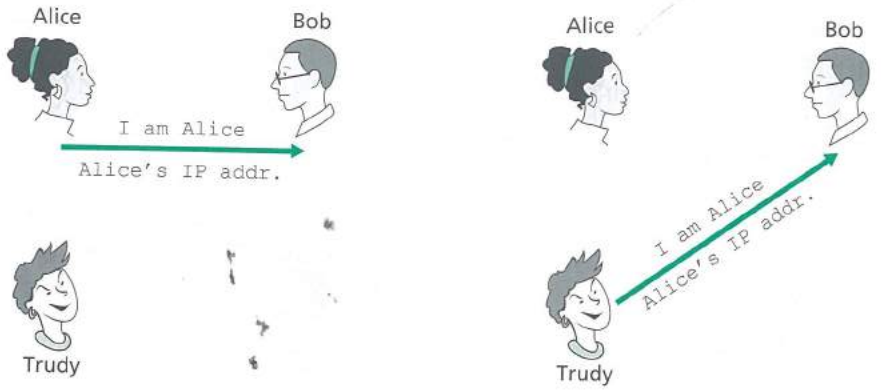


Figure 8.16 ♦ Protocol *ap2.0* and a failure scenario

on, the incorrectly source-addressed datagram would be dutifully forwarded to Bob. This approach, shown in Figure 8.16, is a form of IP spoofing. IP spoofing can be avoided if Trudy's first-hop router is configured to forward only datagrams containing Trudy's IP source address [RFC 2827]. However, this capability is not universally deployed or enforced. Bob would thus be foolish to assume that Trudy's network manager (who might be Trudy herself) had configured Trudy's first-hop router to forward only appropriately addressed datagrams.

8.4.3 Authentication Protocol *ap3.0*

One classic approach to authentication is to use a secret password. The password is a shared secret between the authenticator and the person being authenticated. Gmail, Facebook, telnet, FTP, and many other services use password authentication. In protocol ap3.0, Alice thus sends her secret password to Bob, as shown in Figure 8.17.

Since passwords are so widely used, we might suspect that protocol *ap3.0* is fairly secure. If so, we'd be wrong! The security flaw here is clear. If Trudy eavesdrops on Alice's communication, then she can learn Alice's password. Lest you think this is unlikely, consider the fact that when you Telnet to another machine and log in, the login password is sent unencrypted to the Telnet server. Someone connected to the Telnet client or server's LAN can possibly sniff (read and store) all packets transmitted on the LAN and thus steal the login password. In fact, this is a well-known approach for stealing passwords (see, for example, [Jimenez 1997]). Such a threat is obviously very real, so *ap3.0* clearly won't do.

8.4.4 Authentication Protocol *ap3.1*

Our next idea for fixing ap3.0 is naturally to encrypt the password. By encrypting the password, we can prevent Trudy from learning Alice's password. If we assume

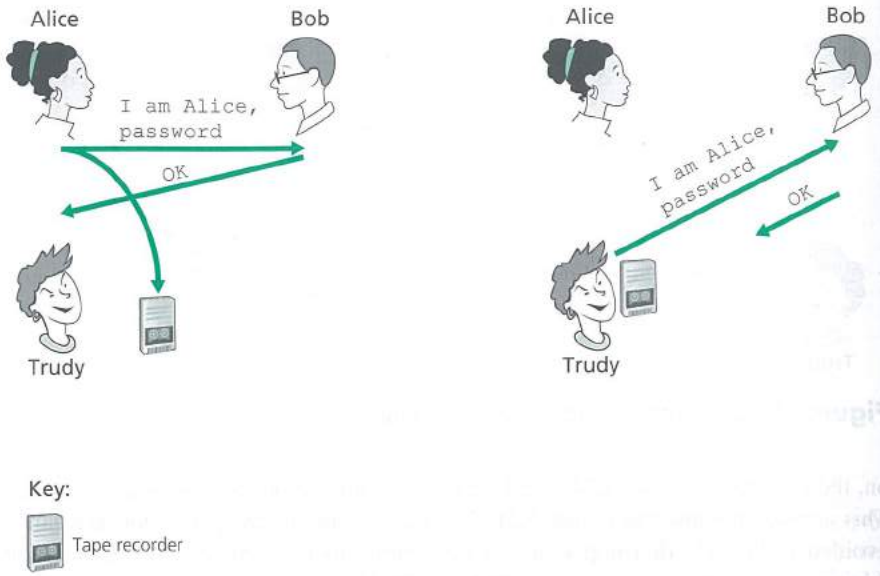


Figure 8.17 ♦ Protocol *ap3.0* and a failure scenario

that Alice and Bob share a symmetric secret key, K_{A-B}, then Alice can encrypt the password and send her identification message, "`I am Alice`," and her encrypted password to Bob. Bob then decrypts the password and, assuming the password is correct, authenticates Alice. Bob feels comfortable in authenticating Alice since Alice not only knows the password, but also knows the shared secret key value needed to encrypt the password. Let's call this protocol *ap3.1*.

While it is true that *ap3.1* prevents Trudy from learning Alice's password, the use of cryptography here does not solve the authentication problem. Bob is subject to a **playback attack**: Trudy need only eavesdrop on Alice's communication, record the encrypted version of the password, and play back the encrypted version of the password to Bob to pretend that she is Alice. The use of an encrypted password in *ap3.1* doesn't make the situation manifestly different from that of protocol *ap3.0* in Figure 8.17.

8.4.5 Authentication Protocol *ap4.0*

The failure scenario in Figure 8.17 resulted from the fact that Bob could not distinguish between the original authentication of Alice and the later playback of Alice's original authentication. That is, Bob could not tell if Alice was live (that is, was currently really on the other end of the connection) or whether the messages he was receiving were a recorded playback of a previous authentication of Alice. The very (*very*) observant reader will recall that the three-way TCP handshake protocol needed

to address the same problem—the server side of a TCP connection did not want to accept a connection if the received SYN segment was an old copy (retransmission) of a SYN segment from an earlier connection. How did the TCP server side solve the problem of determining whether the client was really live? It chose an initial sequence number that had not been used in a very long time, sent that number to the client, and then waited for the client to respond with an ACK segment containing that number. We can adopt the same idea here for authentication purposes.

A **nonce** is a number that a protocol will use only once in a lifetime. That is, once a protocol uses a nonce, it will never use that number again. Our *ap4.0* protocol uses a nonce as follows:

1. Alice sends the message "`I am Alice`" to Bob.
2. Bob chooses a nonce, R, and sends it to Alice.
3. Alice encrypts the nonce using Alice and Bob's symmetric secret key, K_{A-B}, and sends the encrypted nonce, K_{A-B} (R), back to Bob. As in protocol *ap3.1*, it is the fact that Alice knows K_{A-B} and uses it to encrypt a value that lets Bob know that the message he receives was generated by Alice. The nonce is used to ensure that Alice is live.
4. Bob decrypts the received message. If the decrypted nonce equals the nonce he sent Alice, then Alice is authenticated.

Protocol *ap4.0* is illustrated in Figure 8.18. By using the once-in-a-lifetime value, *R*, and then checking the returned value, K_{A-B} (*R*), Bob can be sure that Alice is both who she says she is (since she knows the secret key value needed to encrypt *R*) and live (since she has encrypted the nonce, *R*, that Bob just created).

The use of a nonce and symmetric key cryptography forms the basis of *ap4.0*. A natural question is whether we can use a nonce and public key cryptography (rather than symmetric key cryptography) to solve the authentication problem. This issue is explored in the problems at the end of the chapter.

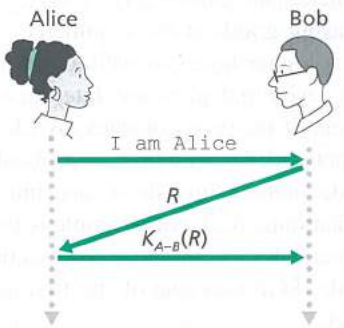


Figure 8.18 ♦ Protocol *ap4.0* and a failure scenario

8.5 Securing E-Mail

In previous sections, we examined fundamental issues in network security, including symmetric key and public key cryptography, end-point authentication, key distribution, message integrity, and digital signatures. We are now going to examine how these tools are being used to provide security in the Internet.

Interestingly, it is possible to provide security services in any of the top four layers of the Internet protocol stack. When security is provided for a specific application-layer protocol, the application using the protocol will enjoy one or more security services, such as confidentiality, authentication, or integrity. When security is provided for a transport-layer protocol, all applications that use that protocol enjoy the security services of the transport protocol. When security is provided at the network layer on a host-to-host basis, all transport-layer segments (and hence all application-layer data) enjoy the security services of the network layer. When security is provided on a link basis, then the data in all frames traveling over the link receive the security services of the link.

In Sections 8.5 through 8.8, we examine how security tools are being used in the application, transport, network, and link layers. Being consistent with the general structure of this book, we begin at the top of the protocol stack and discuss security at the application layer. Our approach is to use a specific application, e-mail, as a case study for application-layer security. We then move down the protocol stack. We'll examine the SSL protocol (which provides security at the transport layer), IPsec (which provides security at the network layer), and the security of the IEEE 802.11 wireless LAN protocol.

You might be wondering why security functionality is being provided at more than one layer in the Internet. Wouldn't it suffice simply to provide the security functionality at the network layer and be done with it? There are two answers to this question. First, although security at the network layer can offer "blanket coverage" by encrypting all the data in the datagrams (that is, all the transport-layer segments) and by authenticating all the source IP addresses, it can't provide user-level security. For example, a commerce site cannot rely on IP-layer security to authenticate a customer who is purchasing goods at the commerce site. Thus, there is a need for security functionality at higher layers as well as blanket coverage at lower layers. Second, it is generally easier to deploy new Internet services, including security services, at the higher layers of the protocol stack. While waiting for security to be broadly deployed at the network layer, which is probably still many years in the future, many application developers "just do it" and introduce security functionality into their favorite applications. A classic example is Pretty Good Privacy (PGP), which provides secure e-mail (discussed later in this section). Requiring only client and server application code, PGP was one of the first security technologies to be broadly used in the Internet.

8.5.1 Secure E-Mail

We now use the cryptographic principles of Sections 8.2 through 8.3 to create a secure e-mail system. We create this high-level design in an incremental manner, at each step introducing new security services. When designing a secure e-mail system, let us keep in mind the racy example introduced in Section 8.1—the love affair between Alice and Bob. Imagine that Alice wants to send an e-mail message to Bob, and Trudy wants to intrude.

Before plowing ahead and designing a secure e-mail system for Alice and Bob, we should consider which security features would be most desirable for them. First and foremost is *confidentiality*. As discussed in Section 8.1, neither Alice nor Bob wants Trudy to read Alice's e-mail message. The second feature that Alice and Bob would most likely want to see in the secure e-mail system is *sender authentication*. In particular, when Bob receives the message "`I don't love you anymore. I never want to see you again. Formerly yours, Alice`," he would naturally want to be sure that the message came from Alice and not from Trudy. Another feature that the two lovers would appreciate is *message integrity*, that is, assurance that the message Alice sends is not modified while en route to Bob. Finally, the e-mail system should provide *receiver authentication*; that is, Alice wants to make sure that she is indeed sending the letter to Bob and not to someone else (for example, Trudy) who is impersonating Bob.

So let's begin by addressing the foremost concern, confidentiality. The most straightforward way to provide confidentiality is for Alice to encrypt the message with symmetric key technology (such as DES or AES) and for Bob to decrypt the message on receipt. As discussed in Section 8.2, if the symmetric key is long enough, and if only Alice and Bob have the key, then it is extremely difficult for anyone else (including Trudy) to read the message. Although this approach is straightforward, it has the fundamental difficulty that we discussed in Section 8.2—distributing a symmetric key so that only Alice and Bob have copies of it. So we naturally consider an alternative approach—public key cryptography (using, for example, RSA). In the public key approach, Bob makes his public key publicly available (e.g., in a public key server or on his personal Web page), Alice encrypts her message with Bob's public key, and she sends the encrypted message to Bob's e-mail address. When Bob receives the message, he simply decrypts it with his private key. Assuming that Alice knows for sure that the public key is Bob's public key, this approach is an excellent means to provide the desired confidentiality. One problem, however, is that public key encryption is relatively inefficient, particularly for long messages.

To overcome the efficiency problem, let's make use of a session key (discussed in Section 8.2.2). In particular, Alice (1) selects a random symmetric session key, K_S, (2) encrypts her message, m, with the symmetric key, (3) encrypts the symmetric key with Bob's public key, K_B^+, (4) concatenates the encrypted message and the encrypted symmetric key to form a "package," and (5) sends the package to Bob's

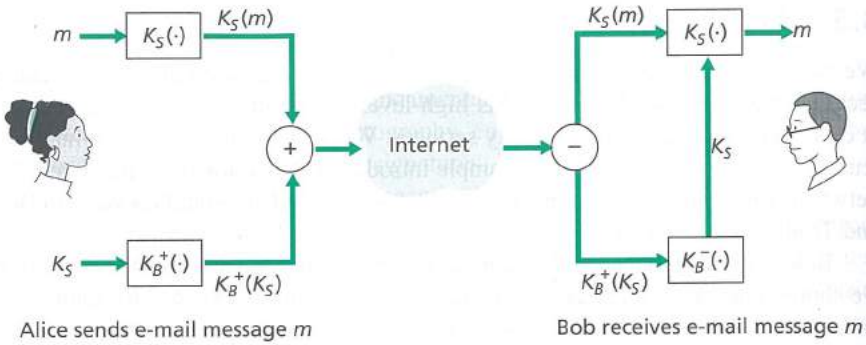


Figure 8.19 ♦ Alice used a symmetric session key, K_S, to send a secret e-mail to Bob

e-mail address. The steps are illustrated in Figure 8.19. (In this and the subsequent figures, the circled "+" represents concatenation and the circled "−" represents deconcatenation.) When Bob receives the package, he (1) uses his private key, K_B^-, to obtain the symmetric key, K_S, and (2) uses the symmetric key K_S to decrypt the message m.

Having designed a secure e-mail system that provides confidentiality, let's now design another system that provides both sender authentication and message integrity. We'll suppose, for the moment, that Alice and Bob are no longer concerned with confidentiality (they want to share their feelings with everyone!), and are concerned only about sender authentication and message integrity. To accomplish this task, we use digital signatures and message digests, as described in Section 8.3. Specifically, Alice (1) applies a hash function, H (for example, MD5), to her message, m, to obtain a message digest, (2) signs the result of the hash function with her private key, K_A^-, to create a digital signature, (3) concatenates the original (unencrypted) message with the signature to create a package, and (4) sends the package to Bob's e-mail address. When Bob receives the package, he (1) applies Alice's public key, K_A^+, to the signed message digest and (2) compares the result of this operation with his own hash, H, of the message. The steps are illustrated in Figure 8.20. As discussed in Section 8.3, if the two results are the same, Bob can be pretty confident that the message came from Alice and is unaltered.

Now let's consider designing an e-mail system that provides confidentiality, sender authentication, *and* message integrity. This can be done by combining the procedures in Figures 8.19 and 8.20. Alice first creates a preliminary package, exactly as in Figure 8.20, that consists of her original message along with a digitally signed hash of the message. She then treats this preliminary package as a message in itself and sends this new message through the sender steps in Figure 8.19, creating a new package that is sent to Bob. The steps applied by Alice are shown in Figure 8.21. When Bob receives the package, he first applies his side of Figure 8.19 and then his

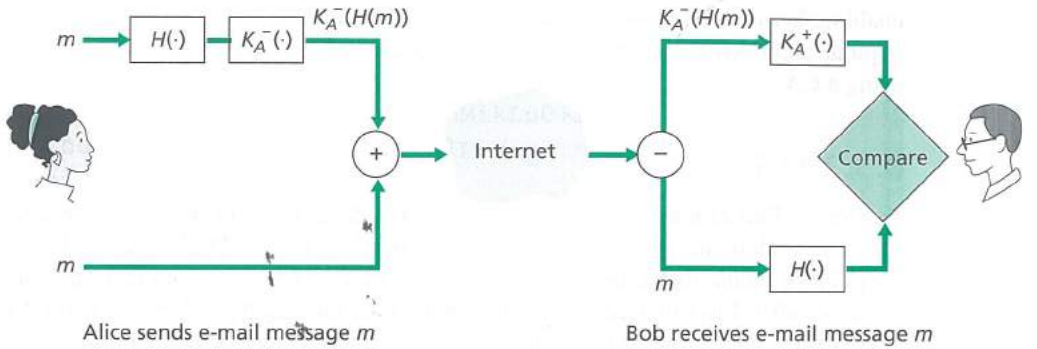


Figure 8.20 ♦ Using hash functions and digital signatures to provide sender authentication and message integrity

side of Figure 8.20. It should be clear that this design achieves the goal of providing confidentiality, sender authentication, and message integrity. Note that, in this scheme, Alice uses public key cryptography twice: once with her own private key and once with Bob's public key. Similarly, Bob also uses public key cryptography twice—once with his private key and once with Alice's public key.

The secure e-mail design outlined in Figure 8.21 probably provides satisfactory security for most e-mail users for most occasions. But there is still one important issue that remains to be addressed. The design in Figure 8.21 requires Alice to obtain Bob's public key, and requires Bob to obtain Alice's public key. The distribution of these public keys is a nontrivial problem. For example, Trudy might masquerade as Bob and give Alice her own public key while saying that it is Bob's public key,

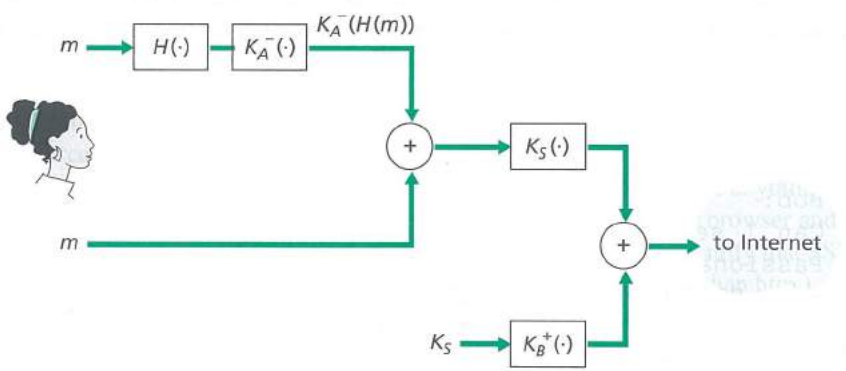


Figure 8.21 ♦ Alice uses symmetric key cyptography, public key cryptography, a hash function, and a digital signature to provide secrecy, sender authentication, and message integrity

enabling her to receive the message meant for Bob. As we learned in Section 8.3, a popular approach for securely distributing public keys is to *certify* the public keys using a CA.

8.5.2 PGP

Written by Phil Zimmermann in 1991, **Pretty Good Privacy (PGP)** is a nice example of an e-mail encryption scheme [PGPI 2016]. Versions of PGP are available in the public domain; for example, you can find the PGP software for your favorite platform as well as lots of interesting reading at the International PGP Home Page [PGPI 2016]. The PGP design is, in essence, the same as the design shown in Figure 8.21. Depending on the version, the PGP software uses MD5 or SHA for calculating the message digest; CAST, triple-DES, or IDEA for symmetric key encryption; and RSA for the public key encryption.

When PGP is installed, the software creates a public key pair for the user. The public key can be posted on the user's Web site or placed in a public key server. The private key is protected by the use of a password. The password has to be entered every time the user accesses the private key. PGP gives the user the option of digitally signing the message, encrypting the message, or both digitally signing and encrypting. Figure 8.22 shows a PGP signed message. This message appears after the MIME header. The encoded data in the message is $K_A^-(H(m))$, that is, the digitally signed message digest. As we discussed above, in order for Bob to verify the integrity of the message, he needs to have access to Alice's public key.

Figure 8.23 shows a secret PGP message. This message also appears after the MIME header. Of course, the plaintext message is not included within the secret e-mail message. When a sender (such as Alice) wants both confidentiality and integrity, PGP contains a message like that of Figure 8.23 within the message of Figure 8.22.

PGP also provides a mechanism for public key certification, but the mechanism is quite different from the more conventional CA. PGP public keys are certified by

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash:  SHA1
Bob:
Can I see you tonight?
Passionately yours, Alice
-----BEGIN PGP SIGNATURE-----
Version: PGP for Personal Privacy 5.0
Charset:  noconv
yhHJRHhGJGhgg/12EpJ+lo8gE4vB3mqJhFEvZP9t6n7G6m5Gw2
-----END PGP SIGNATURE-----
```

Figure 8.22 ♦ A PGP signed message

```
-----BEGIN PGP MESSAGE-----
Version: PGP for Personal Privacy 5.0
u2R4d+/jKmn8Bc5+hgDsqAewsDfrGdszX68liKm5F6Gc4sDfcXyt
RfdS10juHgbcfDssWe7/K=lKhnMikLo0+1/BvcX4t==Ujk9PbcD4
Thdf2awQfgHbnmKlok8iy6gThlp
-----END PGP MESSAGE
```


Figure 8.23 ♦ A secret PGP message

a *web of trust.* Alice herself can certify any key/username pair when she believes the pair really belong together. In addition, PGP permits Alice to say that she trusts another user to vouch for the authenticity of more keys. Some PGP users sign each other's keys by holding key-signing parties. Users physically gather, exchange public keys, and certify each other's keys by signing them with their private keys.

8.6 Securing TCP Connections: SSL

In the previous section, we saw how cryptographic techniques can provide confidentiality, data integrity, and end-point authentication to a specific application, namely, e-mail. In this section, we'll drop down a layer in the protocol stack and examine how cryptography can enhance TCP with security services, including confidentiality, data integrity, and end-point authentication. This enhanced version of TCP is commonly known as **Secure Sockets Layer (SSL)**. A slightly modified version of SSL version 3, called **Transport Layer Security (TLS)**, has been standardized by the IETF [RFC 4346].

The SSL protocol was originally designed by Netscape, but the basic ideas behind securing TCP had predated Netscape's work (for example, see Woo [Woo 1994]). Since its inception, SSL has enjoyed broad deployment. SSL is supported by all popular Web browsers and Web servers, and it is used by Gmail and essentially all Internet commerce sites (including Amazon, eBay, and TaoBao). Hundreds of billions of dollars are spent over SSL every year. In fact, if you have ever purchased anything over the Internet with your credit card, the communication between your browser and the server for this purchase almost certainly went over SSL. (You can identify that SSL is being used by your browser when the URL begins with https: rather than http.)

To understand the need for SSL, let's walk through a typical Internet commerce scenario. Bob is surfing the Web and arrives at the Alice Incorporated site, which is selling perfume. The Alice Incorporated site displays a form in which Bob is supposed to enter the type of perfume and quantity desired, his address, and his payment card number. Bob enters this information, clicks on Submit, and expects to receive (via ordinary postal mail) the purchased perfumes; he also expects to receive

a charge for his order in his next payment card statement. This all sounds good, but if no security measures are taken, Bob could be in for a few surprises.

- If no confidentiality (encryption) is used, an intruder could intercept Bob's order and obtain his payment card information. The intruder could then make purchases at Bob's expense.
- If no data integrity is used, an intruder could modify Bob's order, having him purchase ten times more bottles of perfume than desired.
- Finally, if no server authentication is used, a server could display Alice Incorporated's famous logo when in actuality the site maintained by Trudy, who is masquerading as Alice Incorporated. After receiving Bob's order, Trudy could take Bob's money and run. Or Trudy could carry out an identity theft by collecting Bob's name, address, and credit card number.

SSL addresses these issues by enhancing TCP with confidentiality, data integrity, server authentication, and client authentication.

SSL is often used to provide security to transactions that take place over HTTP. However, because SSL secures TCP, it can be employed by any application that runs over TCP. SSL provides a simple Application Programmer Interface (API) with sockets, which is similar and analogous to TCP's API. When an application wants to employ SSL, the application includes SSL classes/libraries. As shown in Figure 8.24, although SSL technically resides in the application layer, from the developer's perspective it is a transport protocol that provides TCP's services enhanced with security services.

8.6.1 The Big Picture

We begin by describing a simplified version of SSL, one that will allow us to get a big-picture understanding of the *why* and *how* of SSL. We will refer to this simplified

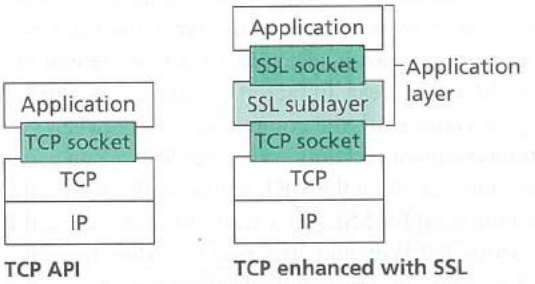


Figure 8.24 ♦ Although SSL technically resides in the application layer, from the developer's perspective it is a transport-layer protocol

version of SSL as "almost-SSL." After describing almost-SSL, in the next subsection we'll then describe the real SSL, filling in the details. Almost-SSL (and SSL) has three phases: *handshake*, *key derivation*, and *data transfer*. We now describe these three phases for a communication session between a client (Bob) and a server (Alice), with Alice having a private/public key pair and a certificate that binds her identity to her public key.

Handshake

During the handshake phase, Bob needs to (a) establish a TCP connection with Alice, (b) verify that Alice is *really* Alice, and (c) send Alice a master secret key, which will be used by both Alice and Bob to generate all the symmetric keys they need for the SSL session. These three steps are shown in Figure 8.25. Note that once the TCP connection is established, Bob sends Alice a hello message. Alice then responds with her certificate, which contains her public key. As discussed in Section 8.3, because the certificate has been certified by a CA, Bob knows for sure that the public key in the certificate belongs to Alice. Bob then generates a Master Secret (MS) (which will only be used for this SSL session), encrypts the MS with Alice's public key to create the Encrypted Master Secret (EMS), and sends the EMS to Alice. Alice decrypts the EMS with her private key to get the MS. After this phase, both Bob and Alice (and no one else) know the master secret for this SSL session.

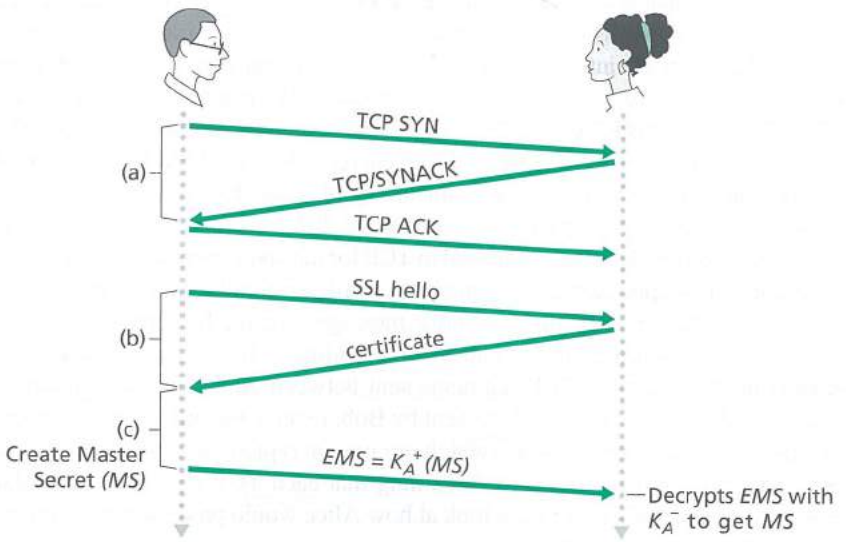


Figure 8.25 ♦ The almost-SSL handshake, beginning with a TCP connection

Key Derivation

In principle, the MS, now shared by Bob and Alice, could be used as the symmetric session key for all subsequent encryption and data integrity checking. It is, however, generally considered safer for Alice and Bob to each use different cryptographic keys, and also to use different keys for encryption and integrity checking. Thus, both Alice and Bob use the MS to generate four keys:

- E_B = session encryption key for data sent from Bob to Alice
- M_B = session MAC key for data sent from Bob to Alice
- E_A = session encryption key for data sent from Alice to Bob
- M_A = session MAC key for data sent from Alice to Bob

Alice and Bob each generate the four keys from the MS. This could be done by simply slicing the MS into four keys. (But in *real* SSL it is a little more complicated, as we'll see.) At the end of the key derivation phase, both Alice and Bob have all four keys. The two encryption keys will be used to encrypt data; the two MAC keys will be used to verify the integrity of the data.

Data Transfer

Now that Alice and Bob share the same four session keys (E_B, M_B, E_A, and M_A), they can start to send secured data to each other over the TCP connection. Since TCP is a byte-stream protocol, a natural approach would be for SSL to encrypt application data on the fly and then pass the encrypted data on the fly to TCP. But if we were to do this, where would we put the MAC for the integrity check? We certainly do not want to wait until the end of the TCP session to verify the integrity of all of Bob's data that was sent over the entire session! To address this issue, SSL breaks the data stream into records, appends a MAC to each record for integrity checking, and then encrypts the record+MAC. To create the MAC, Bob inputs the record data along with the key M_B into a hash function, as discussed in Section 8.3. To encrypt the package record+MAC, Bob uses his session encryption key E_B. This encrypted package is then passed to TCP for transport over the Internet.

Although this approach goes a long way, it still isn't bullet-proof when it comes to providing data integrity for the entire message stream. In particular, suppose Trudy is a woman-in-the-middle and has the ability to insert, delete, and replace segments in the stream of TCP segments sent between Alice and Bob. Trudy, for example, could capture two segments sent by Bob, reverse the order of the segments, adjust the TCP sequence numbers (which are not encrypted), and then send the two reverse-ordered segments to Alice. Assuming that each TCP segment encapsulates exactly one record, let's now take a look at how Alice would process these segments.

1. TCP running in Alice would think everything is fine and pass the two records to the SSL sublayer.
2. SSL in Alice would decrypt the two records.

3. SSL in Alice would use the MAC in each record to verify the data integrity of the two records.
4. SSL would then pass the decrypted byte streams of the two records to the application layer; but the complete byte stream received by Alice would not be in the correct order due to reversal of the records!

You are encouraged to walk through similar scenarios for when Trudy removes segments or when Trudy replays segments.

The solution to this problem, as you probably guessed, is to use sequence numbers. SSL does this as follows. Bob maintains a sequence number counter, which begins at zero and is incremented for each SSL record he sends. Bob doesn't actually include a sequence number in the record itself, but when he calculates the MAC, he includes the sequence number in the MAC calculation. Thus, the MAC is now a hash of the data plus the MAC key M_B *plus the current sequence number*. Alice tracks Bob's sequence numbers, allowing her to verify the data integrity of a record by including the appropriate sequence number in the MAC calculation. This use of SSL sequence numbers prevents Trudy from carrying out a woman-in-the-middle attack, such as reordering or replaying segments. (Why?)

SSL Record

The SSL record (as well as the almost-SSL record) is shown in Figure 8.26. The record consists of a type field, version field, length field, data field, and MAC field. Note that the first three fields are not encrypted. The type field indicates whether the record is a handshake message or a message that contains application data. It is also used to close the SSL connection, as discussed below. SSL at the receiving end uses the length field to extract the SSL records out of the incoming TCP byte stream. The version field is self-explanatory.

8.6.2 A More Complete Picture

The previous subsection covered the almost-SSL protocol; it served to give us a basic understanding of the why and how of SSL. Now that we have a basic understanding of SSL, we can dig a little deeper and examine the essentials of the actual SSL protocol. In parallel to reading this description of the SSL protocol, you are encouraged to complete the Wireshark SSL lab, available at the textbook's Web site.

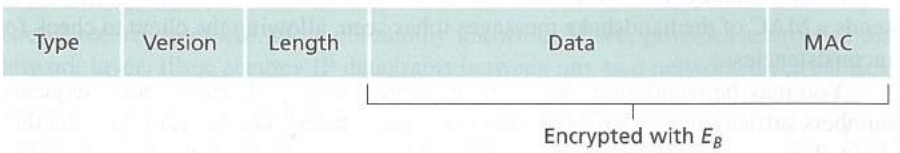


Figure 8.26 ♦ Record format for SSL

SSL Handshake

SSL does not mandate that Alice and Bob use a specific symmetric key algorithm, a specific public-key algorithm, or a specific MAC. Instead, SSL allows Alice and Bob to agree on the cryptographic algorithms at the beginning of the SSL session, during the handshake phase. Additionally, during the handshake phase, Alice and Bob send nonces to each other, which are used in the creation of the session keys (E_B, M_B, E_A, and M_A). The steps of the real SSL handshake are as follows:

1. The client sends a list of cryptographic algorithms it supports, along with a client nonce.
2. From the list, the server chooses a symmetric algorithm (for example, AES), a public key algorithm (for example, RSA with a specific key length), and a MAC algorithm. It sends back to the client its choices, as well as a certificate and a server nonce.
3. The client verifies the certificate, extracts the server's public key, generates a Pre-Master Secret (PMS), encrypts the PMS with the server's public key, and sends the encrypted PMS to the server.
4. Using the same key derivation function (as specified by the SSL standard), the client and server independently compute the Master Secret (MS) from the PMS and nonces. The MS is then sliced up to generate the two encryption and two MAC keys. Furthermore, when the chosen symmetric cipher employs CBC (such as 3DES or AES), then two Initialization Vectors (IVs)—one for each side of the connection—are also obtained from the MS. Henceforth, all messages sent between client and server are encrypted and authenticated (with the MAC).
5. The client sends a MAC of all the handshake messages.
6. The server sends a MAC of all the handshake messages.

The last two steps protect the handshake from tampering. To see this, observe that in step 1, the client typically offers a list of algorithms—some strong, some weak. This list of algorithms is sent in cleartext, since the encryption algorithms and keys have not yet been agreed upon. Trudy, as a woman-in-the-middle, could delete the stronger algorithms from the list, forcing the client to select a weak algorithm. To prevent such a tampering attack, in step 5 the client sends a MAC of the concatenation of all the handshake messages it sent and received. The server can compare this MAC with the MAC of the handshake messages it received and sent. If there is an inconsistency, the server can terminate the connection. Similarly, the server sends a MAC of the handshake messages it has seen, allowing the client to check for inconsistencies.

You may be wondering why there are nonces in steps 1 and 2. Don't sequence numbers suffice for preventing the segment replay attack? The answer is yes, but they don't alone prevent the "connection replay attack." Consider the following connection

play attack. Suppose Trudy sniffs all messages between Alice and Bob. The next y, Trudy masquerades as Bob and sends to Alice exactly the same sequence of essages that Bob sent to Alice on the previous day. If Alice doesn't use nonces, e will respond with exactly the same sequence of messages she sent the previous y. Alice will not suspect any funny business, as each message she receives will ss the integrity check. If Alice is an e-commerce server, she will think that Bob is acing a second order (for exactly the same thing). On the other hand, by including a nce in the protocol, Alice will send different nonces for each TCP session, causing e encryption keys to be different on the two days. Therefore, when Alice receives ayed-back SSL records from Trudy, the records will fail the integrity checks, and e bogus e-commerce transaction will not succeed. In summary, in SSL, nonces are ed to defend against the "connection replay attack" and sequence numbers are used defend against replaying individual packets during an ongoing session.

onnection Closure

t some point, either Bob or Alice will want to end the SSL session. One approach ould be to let Bob end the SSL session by simply terminating the underlying TCP nnection—that is, by having Bob send a TCP FIN segment to Alice. But such a ive design sets the stage for the *truncation attack* whereby Trudy once again gets the middle of an ongoing SSL session and ends the session early with a TCP N. If Trudy were to do this, Alice would think she received all of Bob's data hen actuality she only received a portion of it. The solution to this problem is to dicate in the type field whether the record serves to terminate the SSL session. lthough the SSL type is sent in the clear, it is authenticated at the receiver using the cord's MAC.) By including such a field, if Alice were to receive a TCP FIN before ceiving a closure SSL record, she would know that something funny was going on.

This completes our introduction to SSL. We've seen that it uses many of the yptography principles discussed in Sections 8.2 and 8.3. Readers who want to plore SSL on yet a deeper level can read Rescorla's highly readable book on SSL escorla 2001].

.7 Network-Layer Security: IPsec and Virtual Private Networks

e IP security protocol, more commonly known as **IPsec**, provides security at the twork layer. IPsec secures IP datagrams between any two network-layer entities, cluding hosts and routers. As we will soon describe, many institutions (corporaons, government branches, non-profit organizations, and so on) use IPsec to create **rtual private networks (VPNs)** that run over the public Internet.

Before getting into the specifics of IPsec, let's step back and consider what it means to provide confidentiality at the network layer. With network-layer confidentiality between a pair of network entities (for example, between two routers, between two hosts, or between a router and a host), the sending entity encrypts the payloads of all the datagrams it sends to the receiving entity. The encrypted payload could be a TCP segment, a UDP segment, an ICMP message, and so on. If such a network-layer service were in place, all data sent from one entity to the other—including e-mail, Web pages, TCP handshake messages, and management messages (such as ICMP and SNMP)—would be hidden from any third party that might be sniffing the network. For this reason, network-layer security is said to provide "blanket coverage."

In addition to confidentiality, a network-layer security protocol could potentially provide other security services. For example, it could provide source authentication, so that the receiving entity can verify the source of the secured datagram. A network-layer security protocol could provide data integrity, so that the receiving entity can check for any tampering of the datagram that may have occurred while the datagram was in transit. A network-layer security service could also provide replay-attack prevention, meaning that Bob could detect any duplicate datagrams that an attacker might insert. We will soon see that IPsec indeed provides mechanisms for all these security services, that is, for confidentiality, source authentication, data integrity, and replay-attack prevention.

8.7.1 IPsec and Virtual Private Networks (VPNs)

An institution that extends over multiple geographical regions often desires its own IP network, so that its hosts and servers can send data to each other in a secure and confidential manner. To achieve this goal, the institution could actually deploy a stand-alone physical network—including routers, links, and a DNS infrastructure—that is completely separate from the public Internet. Such a disjoint network, dedicated to a particular institution, is called a **private network**. Not surprisingly, a private network can be very costly, as the institution needs to purchase, install, and maintain its own physical network infrastructure.

Instead of deploying and maintaining a private network, many institutions today create VPNs over the existing public Internet. With a VPN, the institution's inter-office traffic is sent over the public Internet rather than over a physically independent network. But to provide confidentiality, the inter-office traffic is encrypted before it enters the public Internet. A simple example of a VPN is shown in Figure 8.27. Here the institution consists of a headquarters, a branch office, and traveling salespersons that typically access the Internet from their hotel rooms. (There is only one salesperson shown in the figure.) In this VPN, whenever two hosts within headquarters send IP datagrams to each other or whenever two hosts within the branch office want to communicate, they use good-old vanilla IPv4 (that is, without IPsec services). However, when two of the institution's hosts

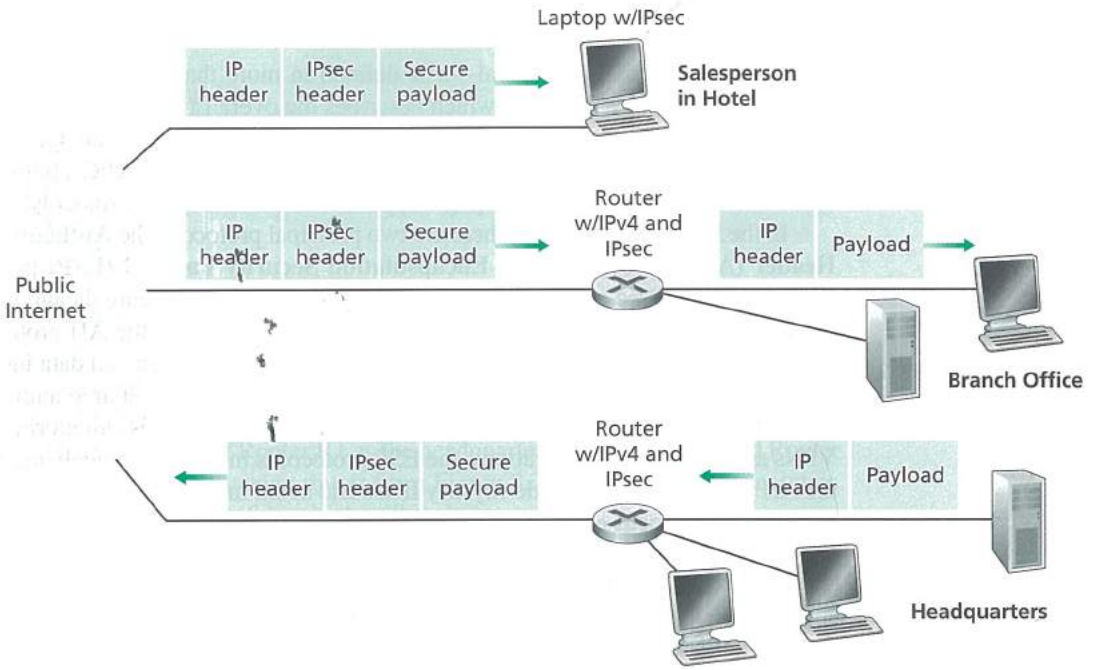


Figure 8.27 ♦ Virtual private network (VPN)

communicate over a path that traverses the public Internet, the traffic is encrypted before it enters the Internet.

To get a feel for how a VPN works, let's walk through a simple example in the context of Figure 8.27. When a host in headquarters sends an IP datagram to a salesperson in a hotel, the gateway router in headquarters converts the vanilla IPv4 datagram into an IPsec datagram and then forwards this IPsec datagram into the Internet. This IPsec datagram actually has a traditional IPv4 header, so that the routers in the public Internet process the datagram as if it were an ordinary IPv4 datagram—to them, the datagram is a perfectly ordinary datagram. But, as shown Figure 8.27, the payload of the IPsec datagram includes an IPsec header, which is used for IPsec processing; furthermore, the payload of the IPsec datagram is encrypted. When the IPsec datagram arrives at the salesperson's laptop, the OS in the laptop decrypts the payload (and provides other security services, such as verifying data integrity) and passes the unencrypted payload to the upper-layer protocol (for example, to TCP or UDP).

We have just given a high-level overview of how an institution can employ IPsec to create a VPN. To see the forest through the trees, we have brushed aside many important details. Let's now take a closer look.

8.7.2 The AH and ESP Protocols

IPsec is a rather complex animal—it is defined in more than a dozen RFCs. Two important RFCs are RFC 4301, which describes the overall IP security architecture, and RFC 6071, which provides an overview of the IPsec protocol suite. Our goal in this textbook, as usual, is not simply to re-hash the dry and arcane RFCs, but instead take a more operational and pedagogic approach to describing the protocols.

In the IPsec protocol suite, there are two principal protocols: the **Authentication Header (AH)** protocol and the **Encapsulation Security Payload (ESP)** protocol. When a source IPsec entity (typically a host or a router) sends secure datagrams to a destination entity (also a host or a router), it does so with either the AH protocol or the ESP protocol. The AH protocol provides source authentication and data integrity but *does not* provide confidentiality. The ESP protocol provides source authentication, data integrity, *and* confidentiality. Because confidentiality is often critical for VPNs and other IPsec applications, the ESP protocol is much more widely used than the AH protocol. In order to de-mystify IPsec and avoid much of its complication, we will henceforth focus exclusively on the ESP protocol. Readers wanting to learn also about the AH protocol are encouraged to explore the RFCs and other online resources.

8.7.3 Security Associations

IPsec datagrams are sent between pairs of network entities, such as between two hosts, between two routers, or between a host and router. Before sending IPsec datagrams from source entity to destination entity, the source and destination entities create a network-layer logical connection. This logical connection is called a **security association (SA)**. An SA is a simplex logical connection; that is, it is unidirectional from source to destination. If both entities want to send secure datagrams to each other, then two SAs (that is, two logical connections) need to be established, one in each direction.

For example, consider once again the institutional VPN in Figure 8.27. This institution consists of a headquarters office, a branch office and, say, n traveling salespersons. For the sake of example, let's suppose that there is bi-directional IPsec traffic between headquarters and the branch office and bi-directional IPsec traffic between headquarters and the salespersons. In this VPN, how many SAs are there? To answer this question, note that there are two SAs between the headquarters gateway router and the branch-office gateway router (one in each direction); for each salesperson's laptop, there are two SAs between the headquarters gateway router and the laptop (again, one in each direction). So, in total, there are $(2 + 2n)$ SAs. *Keep in mind, however, that not all traffic sent into the Internet by the gateway routers or by the laptops will be IPsec secured.* For example, a host in headquarters may want to access a Web server (such as Amazon or Google) in the public Internet. Thus, the gateway router (and the laptops) will emit into the Internet both vanilla IPv4 datagrams and secured IPsec datagrams.

'igure 8.28 ♦ Security association (SA) from R1 to R2

Let's now take a look "inside" an SA. To make the discussion tangible and :oncrete, let's do this in the context of an SA from router R1 to router R2 in Fig-ıre 8.28. (You can think of Router R1 as the headquarters gateway router and Router R2 as the branch office gateway router from Figure 8.27.) Router R1 will maintain tate information about this SA, which will include:

- A 32-bit identifier for the SA, called the **Security Parameter Index (SPI)**
- The origin interface of the SA (in this case 200.168.1.100) and the destination interface of the SA (in this case 193.68.2.23)
- The type of encryption to be used (for example, 3DES with CBC)
- The encryption key
- The type of integrity check (for example, HMAC with MD5)
- The authentication key

Whenever router R1 needs to construct an IPsec datagram for forwarding over his SA, it accesses this state information to determine how it should authenticate ınd encrypt the datagram. Similarly, router R2 will maintain the same state informa-ion for this SA and will use this information to authenticate and decrypt any IPsec latagram that arrives from the SA.

An IPsec entity (router or host) often maintains state information for many SAs. For example, in the VPN example in Figure 8.27 with n salespersons, the headquar-ers gateway router maintains state information for $(2 + 2n)$ SAs. An IPsec entity tores the state information for all of its SAs in its **Security Association Database SAD**), which is a data structure in the entity's OS kernel.

3.7.4 The IPsec Datagram

Having now described SAs, we can now describe the actual IPsec datagram. IPsec ıas two different packet forms, one for the so-called **tunnel mode** and the other for he so-called **transport mode**. The tunnel mode, being more appropriate for VPNs,

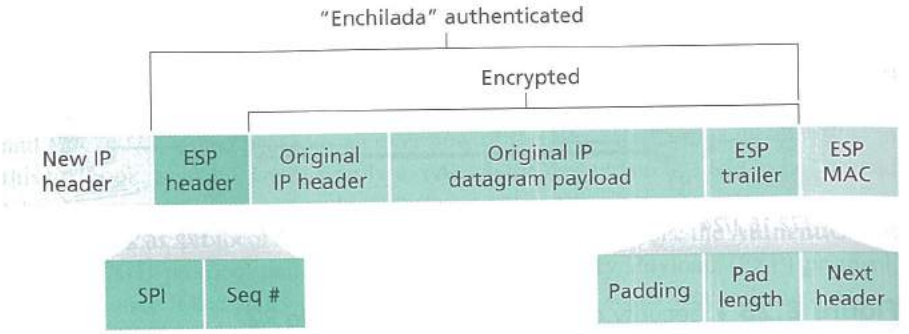


Figure 8.29 ♦ IPsec datagram format

is more widely deployed than the transport mode. In order to further de-mystify IPsec and avoid much of its complication, we henceforth focus exclusively on the tunnel mode. Once you have a solid grip on the tunnel mode, you should be able to easily learn about the transport mode on your own.

The packet format of the IPsec datagram is shown in Figure 8.29. You might think that packet formats are boring and insipid, but we will soon see that the IPsec datagram actually looks and tastes like a popular Tex-Mex delicacy! Let's examine the IPsec fields in the context of Figure 8.28. Suppose router R1 receives an ordinary IPv4 datagram from host 172.16.1.17 (in the headquarters network) which is destined to host 172.16.2.48 (in the branch-office network). Router R1 uses the following recipe to convert this "original IPv4 datagram" into an IPsec datagram:

- Appends to the back of the original IPv4 datagram (which includes the original header fields!) an "ESP trailer" field
- Encrypts the result using the algorithm and key specified by the SA
- Appends to the front of this encrypted quantity a field called "ESP header"; the resulting package is called the "enchilada"
- Creates an authentication MAC over the *whole enchilada* using the algorithm and key specified in the SA
- Appends the MAC to the back of the enchilada forming the *payload*
- Finally, creates a brand new IP header with all the classic IPv4 header fields (together normally 20 bytes long), which it appends before the payload

Note that the resulting IPsec datagram is a bona fide IPv4 datagram, with the traditional IPv4 header fields followed by a payload. But in this case, the payload contains an ESP header, the original IP datagram, an ESP trailer, and an ESP authentication field (with the original datagram and ESP trailer encrypted). The original IP datagram has 172.16.1.17 for the source IP address and 172.16.2.48 for the

estination IP address. Because the IPsec datagram includes the original IP data-
ram, these addresses are included (and encrypted) as part of the payload of the
Psec packet. But what about the source and destination IP addresses that are in the
ew IP header, that is, in the left-most header of the IPsec datagram? As you might
xpect, they are set to the source and destination router interfaces at the two ends of
ıe tunnels, namely, 200.168.1.100 and 193.68.2.23. Also, the protocol number in
ıis new IPv4 header field is not set to that of TCP, UDP, or SMTP, but instead to 50,
esignating that this is an IPsec datagram using the ESP protocol.

After R1 sends the IPsec datagram into the public Internet, it will pass through
ıany routers before reaching R2. Each of these routers will process the datagram as if it
ere an ordinary datagram—they are completely oblivious to the fact that the datagram
; carrying IPsec-encrypted data. For these public Internet routers, because the destina-
on IP address in the outer header is R2, the ultimate destination of the datagram is R2.

Having walked through an example of how an IPsec datagram is constructed,
et's now take a closer look at the ingredients in the enchilada. We see in Figure 8.29
ıat the ESP trailer consists of three fields: padding; pad length; and next header.
ecall that block ciphers require the message to be encrypted to be an integer mul-
ple of the block length. Padding (consisting of meaningless bytes) is used so that
hen added to the original datagram (along with the pad length and next header
elds), the resulting "message" is an integer number of blocks. The pad-length field
ıdicates to the receiving entity how much padding was inserted (and thus needs to
e removed). The next header identifies the type (e.g., UDP) of data contained in the
ayload-data field. The payload data (typically the original IP datagram) and the ESP
ailer are concatenated and then encrypted.

Appended to the front of this encrypted unit is the ESP header, which is sent in
ıe clear and consists of two fields: the SPI and the sequence number field. The SPI
ıdicates to the receiving entity the SA to which the datagram belongs; the receiving
ntity can then index its SAD with the SPI to determine the appropriate authentica-
on/decryption algorithms and keys. The sequence number field is used to defend
gainst replay attacks.

The sending entity also appends an authentication MAC. As stated earlier, the
ending entity calculates a MAC over the whole enchilada (consisting of the ESP
eader, the original IP datagram, and the ESP trailer—with the datagram and trailer
eing encrypted). Recall that to calculate a MAC, the sender appends a secret MAC
ey to the enchilada and then calculates a fixed-length hash of the result.

When R2 receives the IPsec datagram, R2 observes that the destination IP
ddress of the datagram is R2 itself. R2 therefore processes the datagram. Because
ıe protocol field (in the left-most IP header) is 50, R2 sees that it should apply
Psec ESP processing to the datagram. First, peering into the enchilada, R2 uses the
PI to determine to which SA the datagram belongs. Second, it calculates the MAC
f the enchilada and verifies that the MAC is consistent with the value in the ESP
IAC field. If it is, it knows that the enchilada comes from R1 and has not been tam-
ered with. Third, it checks the sequence-number field to verify that the datagram is

fresh (and not a replayed datagram). Fourth, it decrypts the encrypted unit using the decryption algorithm and key associated with the SA. Fifth, it removes padding and extracts the original, vanilla IP datagram. And finally, sixth, it forwards the original datagram into the branch office network toward its ultimate destination. Whew, what a complicated recipe, huh? Well no one ever said that preparing and unraveling an enchilada was easy!

There is actually another important subtlety that needs to be addressed. It centers on the following question: When R1 receives an (unsecured) datagram from a host in the headquarters network, and that datagram is destined to some destination IP address outside of headquarters, how does R1 know whether it should be converted to an IPsec datagram? And if it is to be processed by IPsec, how does R1 know which SA (of many SAs in its SAD) should be used to construct the IPsec datagram? The problem is solved as follows. Along with a SAD, the IPsec entity also maintains another data structure called the **Security Policy Database (SPD)**. The SPD indicates what types of datagrams (as a function of source IP address, destination IP address, and protocol type) are to be IPsec processed; and for those that are to be IPsec processed, which SA should be used. In a sense, the information in a SPD indicates "what" to do with an arriving datagram; the information in the SAD indicates "how" to do it.

Summary of IPsec Services

So what services does IPsec provide, exactly? Let us examine these services from the perspective of an attacker, say Trudy, who is a woman-in-the-middle, sitting somewhere on the path between R1 and R2 in Figure 8.28. Assume throughout this discussion that Trudy does not know the authentication and encryption keys used by the SA. What can and cannot Trudy do? First, Trudy cannot see the original datagram. If fact, not only is the data in the original datagram hidden from Trudy, but so is the protocol number, the source IP address, and the destination IP address. For datagrams sent over the SA, Trudy only knows that the datagram originated from some host in 172.16.1.0/24 and is destined to some host in 172.16.2.0/24. She does not know if it is carrying TCP, UDP, or ICMP data; she does not know if it is carrying HTTP, SMTP, or some other type of application data. This confidentiality thus goes a lot farther than SSL. Second, suppose Trudy tries to tamper with a datagram in the SA by flipping some of its bits. When this tampered datagram arrives at R2, it will fail the integrity check (using the MAC), thwarting Trudy's vicious attempts once again. Third, suppose Trudy tries to masquerade as R1, creating a IPsec datagram with source 200.168.1.100 and destination 193.68.2.23. Trudy's attack will be futile, as this datagram will again fail the integrity check at R2. Finally, because IPsec includes sequence numbers, Trudy will not be able create a successful replay attack. In summary, as claimed at the beginning of this section, IPsec provides—between any pair of devices that process packets through the network layer—confidentiality, source authentication, data integrity, and replay-attack prevention.

8.7.5 IKE: Key Management in IPsec

When a VPN has a small number of end points (for example, just two routers as in Figure 8.28), the network administrator can manually enter the SA information (encryption/authentication algorithms and keys, and the SPIs) into the SADs of the endpoints. Such "manual keying" is clearly impractical for a large VPN, which may consist of hundreds or even thousands of IPsec routers and hosts. Large, geographically distributed deployments require an automated mechanism for creating the SAs. IPsec does this with the Internet Key Exchange (IKE) protocol, specified in RFC 5996.

IKE has some similarities with the handshake in SSL (see Section 8.6). Each IPsec entity has a certificate, which includes the entity's public key. As with SSL, the IKE protocol has the two entities exchange certificates, negotiate authentication and encryption algorithms, and securely exchange key material for creating session keys in the IPsec SAs. Unlike SSL, IKE employs two phases to carry out these tasks.

Let's investigate these two phases in the context of two routers, R1 and R2, in Figure 8.28. The first phase consists of two exchanges of message pairs between R1 and R2:

- During the first exchange of messages, the two sides use Diffie-Hellman (see Homework Problems) to create a bi-directional **IKE SA** between the routers. To keep us all confused, this bi-directional IKE SA is entirely different from the IPsec SAs discussed in Sections 8.6.3 and 8.6.4. The IKE SA provides an authenticated and encrypted channel between the two routers. During this first message-pair exchange, keys are established for encryption and authentication for the IKE SA. Also established is a master secret that will be used to compute IPSec SA keys later in phase 2. Observe that during this first step, RSA public and private keys are not used. In particular, neither R1 nor R2 reveals its identity by signing a message with its private key.
- During the second exchange of messages, both sides reveal their identity to each other by signing their messages. However, the identities are not revealed to a passive sniffer, since the messages are sent over the secured IKE SA channel. Also during this phase, the two sides negotiate the IPsec encryption and authentication algorithms to be employed by the IPsec SAs.

In phase 2 of IKE, the two sides create an SA in each direction. At the end of phase 2, the encryption and authentication session keys are established on both sides for the two SAs. The two sides can then use the SAs to send secured datagrams, as described in Sections 8.7.3 and 8.7.4. The primary motivation for having two phases in IKE is computational cost—since the second phase doesn't involve any public-key cryptography, IKE can generate a large number of SAs between the two IPsec entities with relatively little computational cost.

8.8 Securing Wireless LANs

Security is a particularly important concern in wireless networks, where radio waves carrying frames can propagate far beyond the building containing the wireless base station and hosts. In this section we present a brief introduction to wireless security. For a more in-depth treatment, see the highly readable book by Edney and Arbaugh [Edney 2003].

The issue of security in 802.11 has attracted considerable attention in both technical circles and in the media. While there has been considerable discussion, there has been little debate—there seems to be universal agreement that the original 802.11 specification contains a number of serious security flaws. Indeed, public domain software can now be downloaded that exploits these holes, making those who use the vanilla 802.11 security mechanisms as open to security attacks as users who use no security features at all.

In the following section, we discuss the security mechanisms initially standardized in the 802.11 specification, known collectively as **Wired Equivalent Privacy (WEP)**. As the name suggests, WEP is meant to provide a level of security similar to that found in wired networks. We'll then discuss a few of the security holes in WEP and discuss the 802.11i standard, a fundamentally more secure version of 802.11 adopted in 2004.

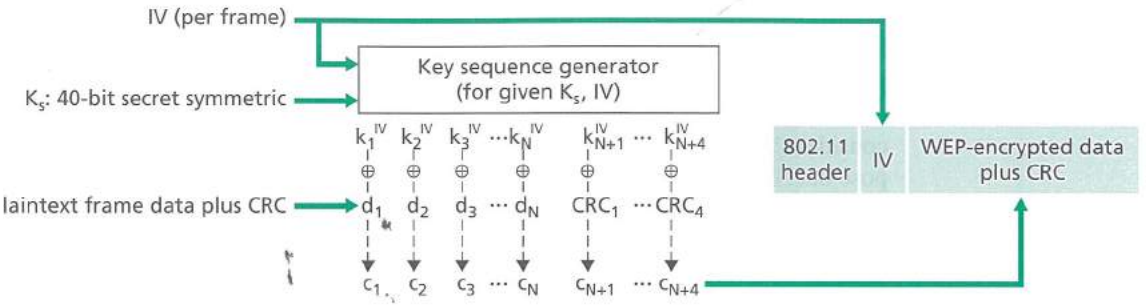
8.8.1 Wired Equivalent Privacy (WEP)

The IEEE 802.11 WEP protocol was designed in 1999 to provide authentication and data encryption between a host and a wireless access point (that is, base station) using a symmetric shared key approach. WEP does not specify a key management algorithm, so it is assumed that the host and wireless access point have somehow agreed on the key via an out-of-band method. Authentication is carried out as follows:

1. A wireless host requests authentication by an access point.
2. The access point responds to the authentication request with a 128-byte nonce value.
3. The wireless host encrypts the nonce using the symmetric key that it shares with the access point.
4. The access point decrypts the host-encrypted nonce.

If the decrypted nonce matches the nonce value originally sent to the host, then the host is authenticated by the access point.

The WEP data encryption algorithm is illustrated in Figure 8.30. A secret 40-bit symmetric key, K_S, is assumed to be known by both a host and the access point. In addition, a 24-bit Initialization Vector (IV) is appended to the 40-bit key to create a 64-bit key that will be used to encrypt a single frame. The IV will



'igure 8.30 ♦ 802.11 WEP protocol

hange from one frame to another, and hence each frame will be encrypted with different 64-bit key. Encryption is performed as follows. First a 4-byte CRC 'alue (see Section 6.2) is computed for the data payload. The payload and the four 'RC bytes are then encrypted using the RC4 stream cipher. We will not cover he details of RC4 here (see [Schneier 1995] and [Edney 2003] for details). For ur purposes, it is enough to know that when presented with a key value (in this ase, the 64-bit (K_S, IV) key), the RC4 algorithm produces a stream of key values, $_1^{IV}$, k_2^{IV}, k_3^{IV}, . . . that are used to encrypt the data and CRC value in a frame. For ractical purposes, we can think of these operations being performed a byte at a ime. Encryption is performed by XOR-ing the ith byte of data, d_i, with the ith key, $_i^{IV}$, in the stream of key values generated by the (K_S, IV) pair to produce the ith yte of ciphertext, c_i:

$$c_i = d_i \oplus k_i^{IV}$$

The IV value changes from one frame to the next and is included *in plaintext* n the header of each WEP-encrypted 802.11 frame, as shown in Figure 8.30. The eceiver takes the secret 40-bit symmetric key that it shares with the sender, appends he IV, and uses the resulting 64-bit key (which is identical to the key used by the ender to perform encryption) to decrypt the frame:

$$d_i = c_i \oplus k_i^{IV}$$

Proper use of the RC4 algorithm requires that the same 64-bit key value *ever* be used more than once. Recall that the WEP key changes on a frame-y-frame basis. For a given K_S (which changes rarely, if ever), this means that here are only 2^{24} unique keys. If these keys are chosen randomly, we can show Edney 2003] that the probability of having chosen the same IV value (and hence sed the same 64-bit key) is more than 99 percent after only 12,000 frames. With Kbyte frame sizes and a data transmission rate of 11 Mbps, only a few seconds are

needed before 12,000 frames are transmitted. Furthermore, since the IV is transmitted in plaintext in the frame, an eavesdropper will know whenever a duplicate IV value is used.

To see one of the several problems that occur when a duplicate key is used, consider the following chosen-plaintext attack taken by Trudy against Alice. Suppose that Trudy (possibly using IP spoofing) sends a request (for example, an HTTP or FTP request) to Alice to transmit a file with known content, $d_1, d_2, d_3, d_4, \ldots$. Trudy also observes the encrypted data $c_1, c_2, c_3, c_4, \ldots$. Since $d_i = c_i \oplus k_i^{IV}$, if we XOR c_i with each side of this equality we have

$$d_i \oplus c_i = k_i^{IV}$$

With this relationship, Trudy can use the known values of d_i and c_i to compute k_i^{IV}. The next time Trudy sees the same value of IV being used, she will know the key sequence $k_1^{IV}, k_2^{IV}, k_3^{IV}, \ldots$ and will thus be able to decrypt the encrypted message.

There are several additional security concerns with WEP as well. [Fluhrer 2001] described an attack exploiting a known weakness in RC4 when certain weak keys are chosen. [Stubblefield 2002] discusses efficient ways to implement and exploit this attack. Another concern with WEP involves the CRC bits shown in Figure 8.30 and transmitted in the 802.11 frame to detect altered bits in the payload. However, an attacker who changes the encrypted content (e.g., substituting gibberish for the original encrypted data), computes a CRC over the substituted gibberish, and places the CRC into a WEP frame can produce an 802.11 frame that will be accepted by the receiver. What is needed here are message integrity techniques such as those we studied in Section 8.3 to detect content tampering or substitution. For more details of WEP security, see [Edney 2003; Wright 2015] and the references therein.

8.8.2 IEEE 802.11i

Soon after the 1999 release of IEEE 802.11, work began on developing a new and improved version of 802.11 with stronger security mechanisms. The new standard, known as 802.11i, underwent final ratification in 2004. As we'll see, while WEP provided relatively weak encryption, only a single way to perform authentication, and no key distribution mechanisms, IEEE 802.11i provides for much stronger forms of encryption, an extensible set of authentication mechanisms, and a key distribution mechanism. In the following, we present an overview of 802.11i; an excellent (streaming audio) technical overview of 802.11i is [TechOnline 2012].

Figure 8.31 overviews the 802.11i framework. In addition to the wireless client and access point, 802.11i defines an authentication server with which the AP can communicate. Separating the authentication server from the AP allows one authentication server to serve many APs, centralizing the (often sensitive) decisions

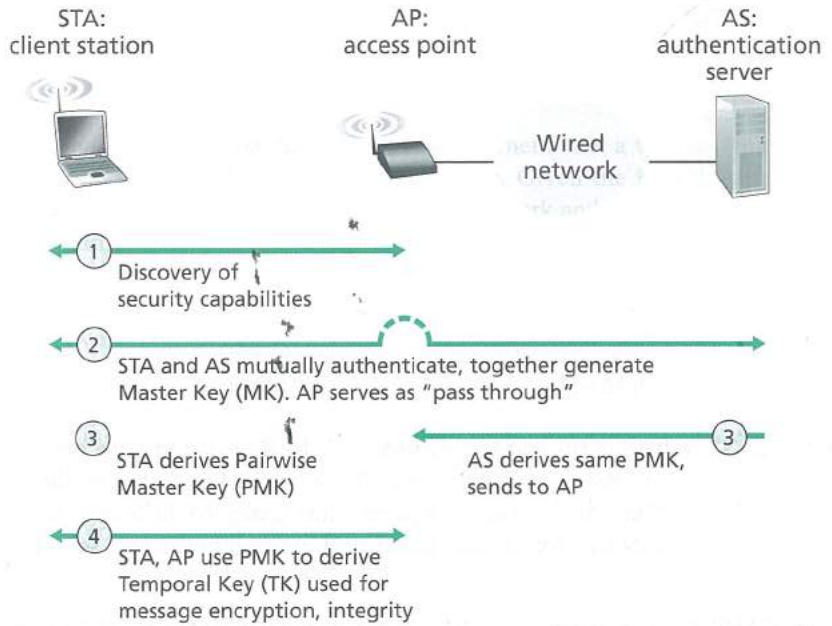


Figure 8.31 ♦ 802.11i: Four phases of operation

regarding authentication and access within the single server, and keeping AP costs and complexity low. 802.11i operates in four phases:

1. *Discovery.* In the discovery phase, the AP advertises its presence and the forms of authentication and encryption that can be provided to the wireless client node. The client then requests the specific forms of authentication and encryption that it desires. Although the client and AP are already exchanging messages, the client has not yet been authenticated nor does it have an encryption key, and so several more steps will be required before the client can communicate with an arbitrary remote host over the wireless channel.
2. *Mutual authentication and Master Key (MK) generation.* Authentication takes place between the wireless client and the authentication server. In this phase, the access point acts essentially as a relay, forwarding messages between the client and the authentication server. The **Extensible Authentication Protocol (EAP)** [RFC 3748] defines the end-to-end message formats used in a simple request/response mode of interaction between the client and authentication server. As shown in Figure 8.32, EAP messages are encapsulated using **EAPoL** (EAP over LAN, [IEEE 802.1X]) and sent over the 802.11 wireless link. These EAP messages are then decapsulated at the access point, and then re-encapsulated using the **RADIUS** protocol for transmission over UDP/IP to the authentication server. While

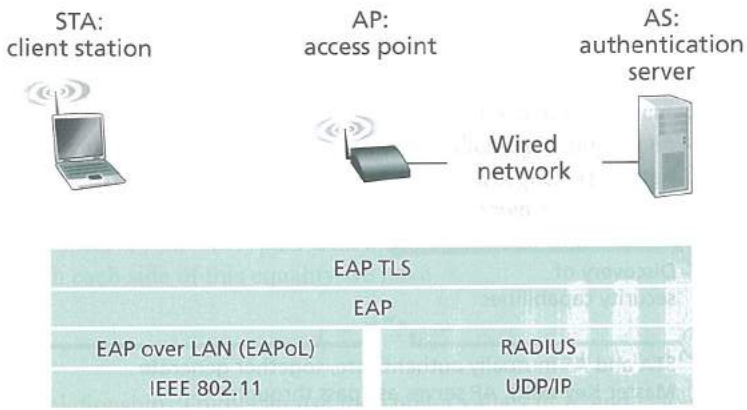


Figure 8.32 ♦ EAP is an end-to-end protocol. EAP messages are encapsulated using EAPoL over the wireless link between the client and the access point, and using RADIUS over UDP/IP between the access point and the authentication server

the RADIUS server and protocol [RFC 2865] are not required by the 802.11i protocol, they are *de facto* standard components for 802.11i. The recently standardized **DIAMETER** protocol [RFC 3588] is likely to replace RADIUS in the near future.

With EAP, the authentication server can choose one of a number of ways to perform authentication. While 802.11i does not mandate a particular authentication method, the EAP-TLS authentication scheme [RFC 5216] is often used. EAP-TLS uses public key techniques (including nonce encryption and message digests) similar to those we studied in Section 8.3 to allow the client and the authentication server to mutually authenticate each other, and to derive a Master Key (MK) that is known to both parties.

3. *Pairwise Master Key (PMK) generation.* The MK is a shared secret known only to the client and the authentication server, which they each use to generate a second key, the Pairwise Master Key (PMK). The authentication server then sends the PMK to the AP. This is where we wanted to be! The client and AP now have a shared key (recall that in WEP, the problem of key distribution was not addressed at all) and have mutually authenticated each other. They're just about ready to get down to business.
4. *Temporal Key (TK) generation.* With the PMK, the wireless client and AP can now generate additional keys that will be used for communication. Of particular interest is the Temporal Key (TK), which will be used to perform the link-level encryption of data sent over the wireless link and to an arbitrary remote host.

802.11i provides several forms of encryption, including an AES-based encryption scheme and a strengthened version of WEP encryption.

8.9 Operational Security: Firewalls and Intrusion Detection Systems

We've seen throughout this chapter that the Internet is not a very safe place—bad guys are out there, wreaking all sorts of havoc. Given the hostile nature of the Internet, let's now consider an organization's network and the network administrator who administers it. From a network administrator's point of view, the world divides quite neatly into two camps—the good guys (who belong to the organization's network, and who should be able to access resources inside the organization's network in a relatively unconstrained manner) and the bad guys (everyone else, whose access to network resources must be carefully scrutinized). In many organizations, ranging from medieval castles to modern corporate office buildings, there is a single point of entry/exit where both good guys and bad guys entering and leaving the organization are security-checked. In a castle, this was done at a gate at one end of the drawbridge; in a corporate building, this is done at the security desk. In a computer network, when traffic entering/leaving a network is security-checked, logged, dropped, or forwarded, it is done by operational devices known as firewalls, intrusion detection systems (IDSs), and intrusion prevention systems (IPSs).

8.9.1 Firewalls

A **firewall** is a combination of hardware and software that isolates an organization's internal network from the Internet at large, allowing some packets to pass and blocking others. A firewall allows a network administrator to control access between the outside world and resources within the administered network by managing the traffic flow to and from these resources. A firewall has three goals:

- *All traffic from outside to inside, and vice versa, passes through the firewall.* Figure 8.33 shows a firewall, sitting squarely at the boundary between the administered network and the rest of the Internet. While large organizations may use multiple levels of firewalls or distributed firewalls [Skoudis 2006], locating a firewall at a single access point to the network, as shown in Figure 8.33, makes it easier to manage and enforce a security-access policy.
- *Only authorized traffic, as defined by the local security policy, will be allowed to pass.* With all traffic entering and leaving the institutional network passing through the firewall, the firewall can restrict access to authorized traffic.
- *The firewall itself is immune to penetration.* The firewall itself is a device connected to the network. If not designed or installed properly, it can be compromised, in which case it provides only a false sense of security (which is worse than no firewall at all!).

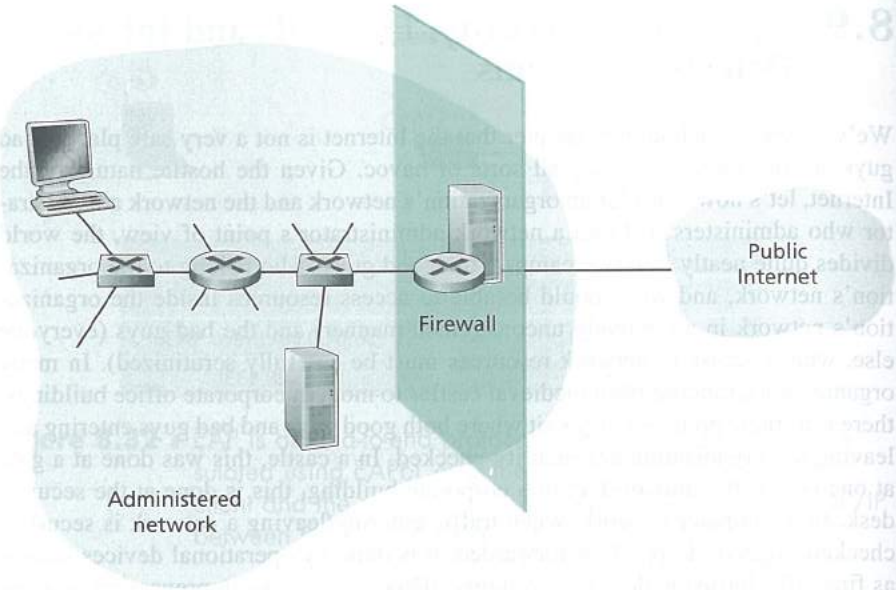


Figure 8.33 ♦ Firewall placement between the administered network and the outside world

Cisco and Check Point are two of the leading firewall vendors today. You can also easily create a firewall (packet filter) from a Linux box using iptables (public-domain software that is normally shipped with Linux). Furthermore, as discussed in Chapters 4 and 5, firewalls are now frequently implemented in routers and controlled remotely using SDNs.

Firewalls can be classified in three categories: **traditional packet filters**, **stateful filters**, and **application gateways**. We'll cover each of these in turn in the following subsections.

Traditional Packet Filters

As shown in Figure 8.33, an organization typically has a gateway router connecting its internal network to its ISP (and hence to the larger public Internet). All traffic leaving and entering the internal network passes through this router, and it is at this router where **packet filtering** occurs. A packet filter examines each datagram in isolation, determining whether the datagram should be allowed to pass or should be dropped based on administrator-specific rules. Filtering decisions are typically based on:

- IP source or destination address
- Protocol type in IP datagram field: TCP, UDP, ICMP, OSPF, and so on
- TCP or UDP source and destination port

Policy	Firewall Setting
No outside Web access.	Drop all outgoing packets to any IP address, port 80.
No incoming TCP connections, except those for organization's public Web server only.	Drop all incoming TCP SYN packets to any IP except 130.207.244.203, port 80.
Prevent Web-radios from eating up the available bandwidth.	Drop all incoming UDP packets—except DNS packets.
Prevent your network from being used for a smurf DoS attack.	Drop all ICMP ping packets going to a "broadcast" address (eg 130.207.255.255).
Prevent your network from being tracerouted.	Drop all outgoing ICMP TTL expired traffic.

Table 8.5 ♦ Policies and corresponding filtering rules for an organization's network 130.207/16 with Web server at 130.207.244.203

- TCP flag bits: SYN, ACK, and so on
- ICMP message type
- Different rules for datagrams leaving and entering the network
- Different rules for the different router interfaces

A network administrator configures the firewall based on the policy of the organization. The policy may take user productivity and bandwidth usage into account as well as the security concerns of an organization. Table 8.5 lists a number of possible polices an organization may have, and how they would be addressed with a packet filter. For example, if the organization doesn't want any incoming TCP connections except those for its public Web server, it can block all incoming TCP SYN segments except TCP SYN segments with destination port 80 and the destination IP address corresponding to the Web server. If the organization doesn't want its users to monopolize access bandwidth with Internet radio applications, it can block all not-critical UDP traffic (since Internet radio is often sent over UDP). If the organization doesn't want its internal network to be mapped (tracerouted) by an outsider, it can block all ICMP TTL expired messages leaving the organization's network.

A filtering policy can be based on a combination of addresses and port numbers. For example, a filtering router could forward all Telnet datagrams (those with a port number of 23) except those going to and coming from a list of specific IP addresses. This policy permits Telnet connections to and from hosts on the allowed list. Unfortunately, basing the policy on external addresses provides no protection against datagrams that have had their source addresses spoofed.

Filtering can also be based on whether or not the TCP ACK bit is set. This trick is quite useful if an organization wants to let its internal clients connect to external servers but wants to prevent external clients from connecting to internal servers.

action	source address	dest address	protocol	source port	dest port	flag bit
allow	222.22/16	outside of 222.22/16	TCP	> 1023	80	any
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK
allow	222.22/16	outside of 222.22/16	UDP	> 1023	53	—
allow	outside of 222.22/16	222.22/16	UDP	53	> 1023	—
deny	all	all	all	all	all	all

Table 8.6 ♦ An access control list for a router interface

Recall from Section 3.5 that the first segment in every TCP connection has the ACK bit set to 0, whereas all the other segments in the connection have the ACK bit set to 1. Thus, if an organization wants to prevent external clients from initiating connections to internal servers, it simply filters all incoming segments with the ACK bit set to 0. This policy kills all TCP connections originating from the outside, but permits connections originating internally.

Firewall rules are implemented in routers with access control lists, with each router interface having its own list. An example of an access control list for an organization 222.22/16 is shown in Table 8.6. This access control list is for an interface that connects the router to the organization's external ISPs. Rules are applied to each datagram that passes through the interface from top to bottom. The first two rules together allow internal users to surf the Web: The first rule allows any TCP packet with destination port 80 to leave the organization's network; the second rule allows any TCP packet with source port 80 and the ACK bit set to enter the organization's network. Note that if an external source attempts to establish a TCP connection with an internal host, the connection will be blocked, even if the source or destination port is 80. The second two rules together allow DNS packets to enter and leave the organization's network. In summary, this rather restrictive access control list blocks all traffic except Web traffic initiated from within the organization and DNS traffic. [CERT Filtering 2012] provides a list of recommended port/protocol packet filterings to avoid a number of well-known security holes in existing network applications.

Stateful Packet Filters

In a traditional packet filter, filtering decisions are made on each packet in isolation. Stateful filters actually track TCP connections, and use this knowledge to make filtering decisions.

source address	dest address	source port	dest port
222.22.1.7	37.96.87.123	12699	80
222.22.93.2	199.1.205.23	37654	80
222.22.65.143	203.77.240.43	48712	80

Table 8.7 ♦ Connection table for stateful filter

To understand stateful filters, let's reexamine the access control list in Table 8.6. Although rather restrictive, the access control list in Table 8.6 nevertheless allows any packet arriving from the outside with ACK = 1 and source port 80 to get through the filter. Such packets could be used by attackers in attempts to crash internal systems with malformed packets, carry out denial-of-service attacks, or map the internal network. The naive solution is to block TCP ACK packets as well, but such an approach would prevent the organization's internal users from surfing the Web.

Stateful filters solve this problem by tracking all ongoing TCP connections in a connection table. This is possible because the firewall can observe the beginning of a new connection by observing a three-way handshake (SYN, SYNACK, and ACK); and it can observe the end of a connection when it sees a FIN packet for the connection. The firewall can also (conservatively) assume that the connection is over when it hasn't seen any activity over the connection for, say, 60 seconds. An example connection table for a firewall is shown in Table 8.7. This connection table indicates that there are currently three ongoing TCP connections, all of which have been initiated from within the organization. Additionally, the stateful filter includes a new column, "check connection," in its access control list, as shown in Table 8.8. Note that Table 8.8 is identical to the access control list in Table 8.6, except now it indicates that the connection should be checked for two of the rules.

Let's walk through some examples to see how the connection table and the extended access control list work hand-in-hand. Suppose an attacker attempts to send a malformed packet into the organization's network by sending a datagram with TCP source port 80 and with the ACK flag set. Further suppose that this packet has source port number 12543 and source IP address 150.23.23.155. When this packet reaches the firewall, the firewall checks the access control list in Table 8.7, which indicates that the connection table must also be checked before permitting this packet to enter the organization's network. The firewall duly checks the connection table, sees that this packet is not part of an ongoing TCP connection, and rejects the packet. As a second example, suppose that an internal user wants to surf an external Web site. Because this user first sends a TCP SYN segment, the user's TCP connection gets recorded in the connection table. When

action	source address	dest address	protocol	source port	dest port	flag bit	check conxion
allow	222.22/16	outside of 222.22/16	TCP	> 1023	80	any	
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK	X
allow	222.22/16	outside of 222.22/16	UDP	> 1023	53	—	
allow	outside of 222.22/16	222.22/16	UDP	53	> 1023	—	X
deny	all	all	all	all	all	all	

Table 8.8 ♦ Access control list for stateful filter

the Web server sends back packets (with the ACK bit necessarily set), the firewall checks the table and sees that a corresponding connection is in progress. The firewall will thus let these packets pass, thereby not interfering with the internal user's Web surfing activity.

Application Gateway

In the examples above, we have seen that packet-level filtering allows an organization to perform coarse-grain filtering on the basis of the contents of IP and TCP/UDP headers, including IP addresses, port numbers, and acknowledgment bits. But what if an organization wants to provide a Telnet service to a restricted set of internal users (as opposed to IP addresses)? And what if the organization wants such privileged users to authenticate themselves first before being allowed to create Telnet sessions to the outside world? Such tasks are beyond the capabilities of traditional and stateful filters. Indeed, information about the identity of the internal users is application-layer data and is not included in the IP/TCP/UDP headers.

To have finer-level security, firewalls must combine packet filters with application gateways. Application gateways look beyond the IP/TCP/UDP headers and make policy decisions based on application data. An **application gateway** is an application-specific server through which all application data (inbound and outbound) must pass. Multiple application gateways can run on the same host, but each gateway is a separate server with its own processes.

To get some insight into application gateways, let's design a firewall that allows only a restricted set of internal users to Telnet outside and prevents all external clients from Telneting inside. Such a policy can be accomplished by implementing

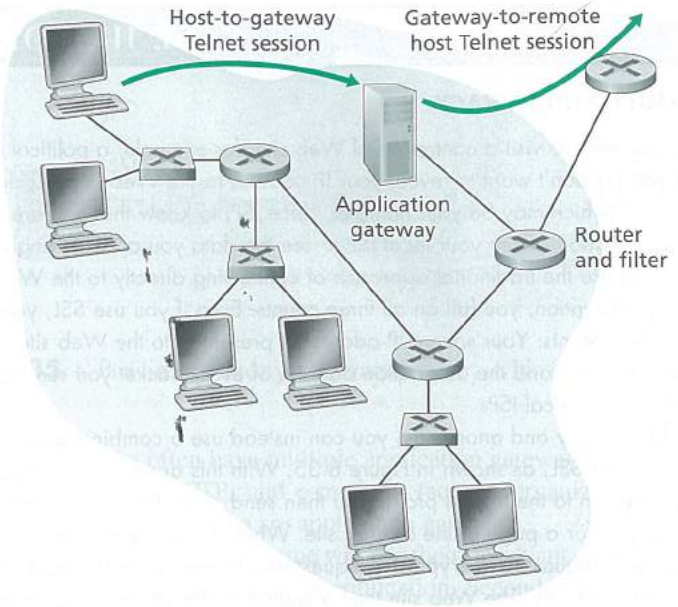


Figure 8.34 ♦ Firewall consisting of an application gateway and a filter

a combination of a packet filter (in a router) and a Telnet application gateway, as shown in Figure 8.34. The router's filter is configured to block all Telnet connections except those that originate from the IP address of the application gateway. Such a filter configuration forces all outbound Telnet connections to pass through the application gateway. Consider now an internal user who wants to Telnet to the outside world. The user must first set up a Telnet session with the application gateway. An application running in the gateway, which listens for incoming Telnet sessions, prompts the user for a user ID and password. When the user supplies this information, the application gateway checks to see if the user has permission to Telnet to the outside world. If not, the Telnet connection from the internal user to the gateway is terminated by the gateway. If the user has permission, then the gateway (1) prompts the user for the host name of the external host to which the user wants to connect, (2) sets up a Telnet session between the gateway and the external host, and (3) relays to the external host all data arriving from the user, and relays to the user all data arriving from the external host. Thus, the Telnet application gateway not only performs user authorization but also acts as a Telnet server and a Telnet client, relaying information between the user and the remote Telnet server. Note that the filter will permit step 2 because the gateway initiates the Telnet connection to the outside world.

CASE HISTORY

ANONYMITY AND PRIVACY

Suppose you want to visit a controversial Web site (for example, a political activist site) and you (1) don't want to reveal your IP address to the Web site, (2) don't want your local ISP (which may be your home or office ISP) to know that you are visiting the site, and (3) don't want your local ISP to see the data you are exchanging with the site. If you use the traditional approach of connecting directly to the Web site without any encryption, you fail on all three counts. Even if you use SSL, you fail on the first two counts: Your source IP address is presented to the Web site in every datagram you send; and the destination address of every packet you send can easily be sniffed by your local ISP.

To obtain privacy and anonymity, you can instead use a combination of a trusted proxy server and SSL, as shown in Figure 8.35. With this approach, you first make an SSL connection to the trusted proxy. You then send, into this SSL connection, an HTTP request for a page at the desired site. When the proxy receives the SSL-encrypted HTTP request, it decrypts the request and forwards the cleartext HTTP request to the Web site. The Web site then responds to the proxy, which in turn forwards the response to you over SSL. Because the Web site only sees the IP address of the proxy, and not of your client's address, you are indeed obtaining anonymous access to the Web site. And because all traffic between you and the proxy is encrypted, your local ISP cannot invade your privacy by logging the site you visited or recording the data you are exchanging. Many companies today (such as proxify.com) make available such proxy services.

Of course, in this solution, your proxy knows everything: It knows your IP address and the IP address of the site you're surfing; and it can see all the traffic in cleartext exchanged between you and the Web site. Such a solution, therefore, is only as good as the trustworthiness of the proxy. A more robust approach, taken by the TOR anonymizing and privacy service, is to route your traffic through a series of non-colluding proxy servers [TOR 2016]. In particular, TOR allows independent individuals to contribute proxies to its proxy pool. When a user connects to a server using TOR, TOR randomly chooses (from its proxy pool) a chain of three proxies and routes all traffic between client and server over the chain. In this manner, assuming the proxies do not collude, no one knows that communication took place between your IP address and the target Web site. Furthermore, although cleartext is sent between the last proxy and the server, the last proxy doesn't know what IP address is sending and receiving the cleartext.

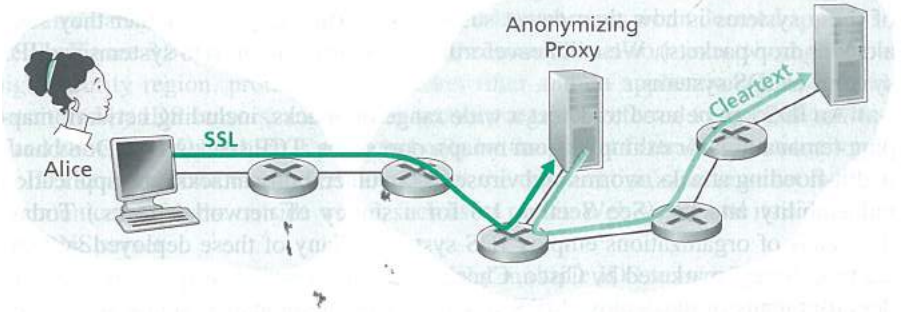


Figure 8.35 ♦ Providing anonymity and privacy with a proxy

Internal networks often have multiple application gateways, for example, gateways for Telnet, HTTP, FTP, and e-mail. In fact, an organization's mail server (see Section 2.3) and Web cache are application gateways.

Application gateways do not come without their disadvantages. First, a different application gateway is needed for each application. Second, there is a performance penalty to be paid, since all data will be relayed via the gateway. This becomes a concern particularly when multiple users or applications are using the same gateway machine. Finally, the client software must know how to contact the gateway when the user makes a request, and must know how to tell the application gateway what external server to connect to.

8.9.2 Intrusion Detection Systems

We've just seen that a packet filter (traditional and stateful) inspects IP, TCP, UDP, and ICMP header fields when deciding which packets to let pass through the firewall. However, to detect many attack types, we need to perform **deep packet inspection**, that is, look beyond the header fields and into the actual application data that the packets carry. As we saw in Section 8.9.1, application gateways often do deep packet inspection. But an application gateway only does this for a specific application.

Clearly, there is a niche for yet another device—a device that not only examines the headers of all packets passing through it (like a packet filter), but also performs deep packet inspection (unlike a packet filter). When such a device observes a suspicious packet, or a suspicious series of packets, it could prevent those packets from entering the organizational network. Or, because the activity is only deemed as suspicious, the device could let the packets pass, but send alerts to a network administrator, who can then take a closer look at the traffic and take appropriate actions. A device that generates alerts when it observes potentially malicious traffic is called an **intrusion detection system (IDS)**. A device that filters out suspicious traffic is called an **intrusion prevention system (IPS)**. In this section we study

both systems—IDS and IPS—together, since the most interesting technical aspect of these systems is how they detect suspicious traffic (and not whether they send alerts or drop packets). We will henceforth collectively refer to IDS systems and IPS systems as IDS systems.

An IDS can be used to detect a wide range of attacks, including network mapping (emanating, for example, from nmap), port scans, TCP stack scans, DoS bandwidth-flooding attacks, worms and viruses, OS vulnerability attacks, and application vulnerability attacks. (See Section 1.6 for a survey of network attacks.) Today, thousands of organizations employ IDS systems. Many of these deployed systems are proprietary, marketed by Cisco, Check Point, and other security equipment vendors. But many of the deployed IDS systems are public-domain systems, such as the immensely popular Snort IDS system (which we'll discuss shortly).

An organization may deploy one or more IDS sensors in its organizational network. Figure 8.36 shows an organization that has three IDS sensors. When multiple sensors are deployed, they typically work in concert, sending information about

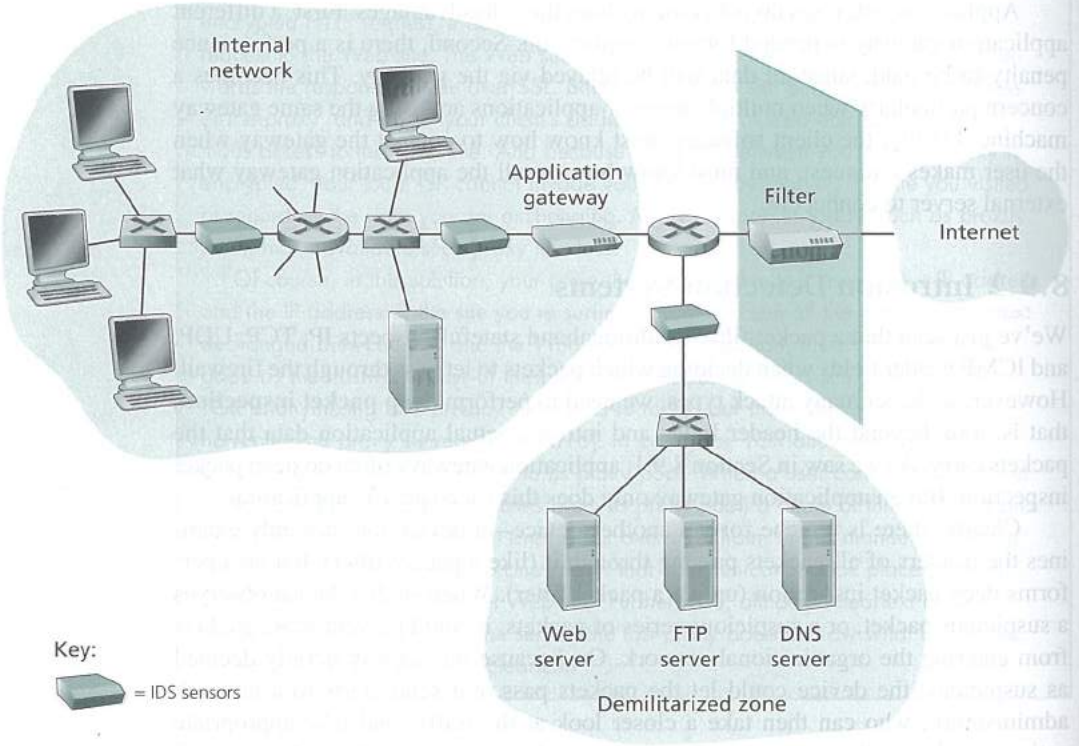


Figure 8.36 ♦ An organization deploying a filter, an application gateway, and IDS sensors

uspicious traffic activity to a central IDS processor, which collects and integrates he information and sends alarms to network administrators when deemed appropri-te. In Figure 8.36, the organization has partitioned its network into two regions: a igh-security region, protected by a packet filter and an application gateway and nonitored by IDS sensors; and a lower-security region—referred to as the **demilita-ized zone (DMZ)**—which is protected only by the packet filter, but also monitored y IDS sensors. Note that the DMZ includes the organization's servers that need to ommunicate with the outside world, such as its public Web server and its authorita-ive DNS server.

You may be wondering at this stage, why multiple IDS sensors? Why not just lace one IDS sensor just behind the packet filter (or even integrated with the packet ilter) in Figure 8.36? We will soon see that an IDS not only needs to do deep packet nspection, but must also compare each passing packet with tens of thousands of 'signatures"; this can be a significant amount of processing, particularly if the organ-zation receives gigabits/sec of traffic from the Internet. By placing the IDS sensors urther downstream, each sensor sees only a fraction of the organization's traffic, nd can more easily keep up. Nevertheless, high-performance IDS and IPS systems re available today, and many organizations can actually get by with just one sensor ocated near its access router.

IDS systems are broadly classified as either **signature-based systems** or **anomaly-based systems**. A signature-based IDS maintains an extensive database of attack ignatures. Each signature is a set of rules pertaining to an intrusion activity. A sig-nature may simply be a list of characteristics about a single packet (e.g., source and lestination port numbers, protocol type, and a specific string of bits in the packet payload), or may relate to a series of packets. The signatures are normally created by killed network security engineers who research known attacks. An organization's etwork administrator can customize the signatures or add its own to the database.

Operationally, a signature-based IDS sniffs every packet passing by it, compar-ng each sniffed packet with the signatures in its database. If a packet (or series of packets) matches a signature in the database, the IDS generates an alert. The alert ould be sent to the network administrator in an e-mail message, could be sent to the etwork management system, or could simply be logged for future inspection.

Signature-based IDS systems, although widely deployed, have a number of limi-ations. Most importantly, they require previous knowledge of the attack to generate n accurate signature. In other words, a signature-based IDS is completely blind to ew attacks that have yet to be recorded. Another disadvantage is that even if a sig-nature is matched, it may not be the result of an attack, so that a false alarm is gener-ted. Finally, because every packet must be compared with an extensive collection f signatures, the IDS can become overwhelmed with processing and actually fail to letect many malicious packets.

An anomaly-based IDS creates a traffic profile as it observes traffic in normal peration. It then looks for packet streams that are statistically unusual, for exam-le, an inordinate percentage of ICMP packets or a sudden exponential growth in

port scans and ping sweeps. The great thing about anomaly-based IDS systems is that they don't rely on previous knowledge about existing attacks—that is, they can potentially detect new, undocumented attacks. On the other hand, it is an extremely challenging problem to distinguish between normal traffic and statistically unusual traffic. To date, most IDS deployments are primarily signature-based, although some include some anomaly-based features.

Snort

Snort is a public-domain, open source IDS with hundreds of thousands of existing deployments [Snort 2012; Koziol 2003]. It can run on Linux, UNIX, and Windows platforms. It uses the generic sniffing interface libpcap, which is also used by Wireshark and many other packet sniffers. It can easily handle 100 Mbps of traffic; for installations with gibabit/sec traffic rates, multiple Snort sensors may be needed.

To gain some insight into Snort, let's take a look at an example of a Snort signature:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any
(msg:"ICMP PING NMAP"; dsize: 0; itype: 8;)
```

This signature is matched by any ICMP packet that enters the organization's network (`$HOME_NET`) from the outside (`$EXTERNAL_NET`), is of type 8 (ICMP ping), and has an empty payload (dsize = 0). Since nmap (see Section 1.6) generates ping packets with these specific characteristics, this signature is designed to detect nmap ping sweeps. When a packet matches this signature, Snort generates an alert that includes the message "`ICMP PING NMAP`".

Perhaps what is most impressive about Snort is the vast community of users and security experts that maintain its signature database. Typically within a few hours of a new attack, the Snort community writes and releases an attack signature, which is then downloaded by the hundreds of thousands of Snort deployments distributed around the world. Moreover, using the Snort signature syntax, network administrators can tailor the signatures to their own organization's needs by either modifying existing signatures or creating entirely new ones.

8.10 Summary

In this chapter, we've examined the various mechanisms that our secret lovers, Bob and Alice, can use to communicate securely. We've seen that Bob and Alice are interested in confidentiality (so they alone are able to understand the contents of a transmitted message), end-point authentication (so they are sure that they are talking

vith each other), and message integrity (so they are sure that their messages are not ıltered in transit). Of course, the need for secure communication is not confined to ;ecret lovers. Indeed, we saw in Sections 8.5 through 8.8 that security can be used in /arious layers in a network architecture to protect against bad guys who have a large ırsenal of possible attacks at hand.

The first part of this chapter presented various principles underlying secure :ommunication. In Section 8.2, we covered cryptographic techniques for encrypting ınd decrypting data, including symmetric key cryptography and public key cryp- ography. DES and RSA were examined as specific case studies of these two major :lasses of cryptographic techniques in use in today's networks.

In Section 8.3, we examined two approaches for providing message integrity: nessage authentication codes (MACs) and digital signatures. The two approaches ıave a number of parallels. Both use cryptographic hash functions and both tech- ıiques enable us to verify the source of the message as well as the integrity of the nessage itself. One important difference is that MACs do not rely on encryption vhereas digital signatures require a public key infrastructure. Both techniques are :xtensively used in practice, as we saw in Sections 8.5 through 8.8. Furthermore, ligital signatures are used to create digital certificates, which are important for veri- 'ying the validity of public keys. In Section 8.4, we examined endpoint authentica- ion and introduced nonces to defend against the replay attack.

In Sections 8.5 through 8.8 we examined several security networking protocols :hat enjoy extensive use in practice. We saw that symmetric key cryptography is at :he core of PGP, SSL, IPsec, and wireless security. We saw that public key cryptog- :aphy is crucial for both PGP and SSL. We saw that PGP uses digital signatures for nessage integrity, whereas SSL and IPsec use MACs. Having now an understand- ıng of the basic principles of cryptography, and having studied how these princi- ɔles are actually used, you are now in position to design your own secure network ɔrotocols!

Armed with the techniques covered in Sections 8.2 through 8.8, Bob and Alice :an communicate securely. (One can only hope that they are networking students vho have learned this material and can thus avoid having their tryst uncovered by Trudy!) But confidentiality is only a small part of the network security picture. As we learned in Section 8.9, increasingly, the focus in network security has been on securing the network infrastructure against a potential onslaught by the bad guys. In the latter part of this chapter, we thus covered firewalls and IDS systems which inspect packets entering and leaving an organization's network.

This chapter has covered a lot of ground, while focusing on the most important topics in modern network security. Readers who desire to dig deeper are encour- aged to investigate the references cited in this chapter. In particular, we recommend [Skoudis 2006] for attacks and operational security, [Kaufman 1995] for cryptog- raphy and how it applies to network security, [Rescorla 2001] for an in-depth but readable treatment of SSL, and [Edney 2003] for a thorough discussion of 802.11 security, including an insightful investigation into WEP and its flaws.

Homework Problems and Questions

Chapter 8 Review Problems

SECTION 8.1

R1. What are the differences between message confidentiality and message integrity? Can you have confidentiality without integrity? Can you have integrity without confidentiality? Justify your answer.

R2. Internet entities (routers, switches, DNS servers, Web servers, user end systems, and so on) often need to communicate securely. Give three specific example pairs of Internet entities that may want secure communication.

SECTION 8.2

R3. From a service perspective, what is an important difference between a symmetric-key system and a public-key system?

R4. Suppose that an intruder has an encrypted message as well as the decrypted version of that message. Can the intruder mount a ciphertext-only attack, a known-plaintext attack, or a chosen-plaintext attack?

R5. Consider an 8-block cipher. How many possible input blocks does this cipher have? How many possible mappings are there? If we view each mapping as a key, then how many possible keys does this cipher have?

R6. Suppose N people want to communicate with each of $N - 1$ other people using symmetric key encryption. All communication between any two people, i and j, is visible to all other people in this group of N, and no other person in this group should be able to decode their communication. How many keys are required in the system as a whole? Now suppose that public key encryption is used. How many keys are required in this case?

R7. Suppose $n = 10{,}000$, $a = 10{,}023$, and $b = 10{,}004$. Use an identity of modular arithmetic to calculate in your head $(a \cdot b) \bmod n$.

R8. Suppose you want to encrypt the message 10101111 by encrypting the decimal number that corresponds to the message. What is the decimal number?

SECTIONS 8.3–8.4

R9. In what way does a hash provide a better message integrity check than a checksum (such as the Internet checksum)?

R10. Can you "decrypt" a hash of a message to get the original message? Explain your answer.

R11. Consider a variation of the MAC algorithm (Figure 8.9) where the sender sends $(m, H(m) + s)$, where $H(m) + s$ is the concatenation of $H(m)$ and s. Is this variation flawed? Why or why not?

R12. What does it mean for a signed document to be verifiable and nonforgeable?

R13. In what way does the public-key encrypted message hash provide a better digital signature than the public-key encrypted message?

R14. Suppose certifier.com creates a certificate for foo.com. Typically, the entire certificate would be encrypted with certifier.com's public key. True or false?

R15. Suppose Alice has a message that she is ready to send to anyone who asks. Thousands of people want to obtain Alice's message, but each wants to be sure of the integrity of the message. In this context, do you think a MAC-based or a digital-signature-based integrity scheme is more suitable? Why?

R16. What is the purpose of a nonce in an end-point authentication protocol?

R17. What does it mean to say that a nonce is a once-in-a-lifetime value? In whose lifetime?

R18. Is the message integrity scheme based on HMAC susceptible to playback attacks? If so, how can a nonce be incorporated into the scheme to remove this susceptibility?

SECTIONS 8.5–8.8

R19. Suppose that Bob receives a PGP message from Alice. How does Bob know for sure that Alice created the message (rather than, say, Trudy)? Does PGP use a MAC for message integrity?

R20. In the SSL record, there is a field for SSL sequence numbers. True or false?

R21. What is the purpose of the random nonces in the SSL handshake?

R22. Suppose an SSL session employs a block cipher with CBC. True or false: The server sends to the client the IV in the clear.

R23. Suppose Bob initiates a TCP connection to Trudy who is pretending to be Alice. During the handshake, Trudy sends Bob Alice's certificate. In what step of the SSL handshake algorithm will Bob discover that he is not communicating with Alice?

R24. Consider sending a stream of packets from Host A to Host B using IPsec. Typically, a new SA will be established for each packet sent in the stream. True or false?

R25. Suppose that TCP is being run over IPsec between headquarters and the branch office in Figure 8.28. If TCP retransmits the same packet, then the two corresponding packets sent by R1 packets will have the same sequence number in the ESP header. True or false?

R26. An IKE SA and an IPsec SA are the same thing. True or false?

R27. Consider WEP for 802.11. Suppose that the data is 10101100 and the keystream is 1111000. What is the resulting ciphertext?

R28. In WEP, an IV is sent in the clear in every frame. True or false?

SECTION 8.9

R29. Stateful packet filters maintain two data structures. Name them and briefly describe what they do.

R30. Consider a traditional (stateless) packet filter. This packet filter may filter packets based on TCP flag bits as well as other header fields. True or false?

R31. In a traditional packet filter, each interface can have its own access control list. True or false?

R32. Why must an application gateway work in conjunction with a router filter to be effective?

R33. Signature-based IDSs and IPSs inspect into the payloads of TCP and UDP segments. True or false?

Problems

P1. Using the monoalphabetic cipher in Figure 8.3, encode the message "This is an easy problem." Decode the message "rmij'u uamu xyj."

P2. Show that Trudy's known-plaintext attack, in which she knows the (ciphertext, plaintext) translation pairs for seven letters, reduces the number of possible substitutions to be checked in the example in Section 8.2.1 by approximately 109.

P3. Consider the polyalphabetic system shown in Figure 8.4. Will a chosen-plaintext attack that is able to get the plaintext encoding of the message "The quick brown fox jumps over the lazy dog." be sufficient to decode all messages? Why or why not?

P4. Consider the block cipher in Figure 8.5. Suppose that each block cipher T_i simply reverses the order of the eight input bits (so that, for example, 11110000 becomes 00001111). Further suppose that the 64-bit scrambler does not modify any bits (so that the output value of the mth bit is equal to the input value of the mth bit). (a) With $n = 3$ and the original 64-bit input equal to 10100000 repeated eight times, what is the value of the output? (b) Repeat part (a) but now change the last bit of the original 64-bit input from a 0 to a 1. (c) Repeat parts (a) and (b) but now suppose that the 64-bit scrambler inverses the order of the 64 bits.

P5. Consider the block cipher in Figure 8.5. For a given "key" Alice and Bob would need to keep eight tables, each 8 bits by 8 bits. For Alice (or Bob) to store all eight tables, how many bits of storage are necessary? How does this number compare with the number of bits required for a full-table 64-bit block cipher?

P6. Consider the 3-bit block cipher in Table 8.1. Suppose the plaintext is 100100100. (a) Initially assume that CBC is not used. What is the resulting ciphertext? (b) Suppose Trudy sniffs the ciphertext. Assuming she knows that

a 3-bit block cipher without CBC is being employed (but doesn't know the specific cipher), what can she surmise? (c) Now suppose that CBC is used with IV = 111. What is the resulting ciphertext?

P7. (a) Using RSA, choose $p = 3$ and $q = 11$, and encode the word "dog" by encrypting each letter separately. Apply the decryption algorithm to the encrypted version to recover the original plaintext message. (b) Repeat part (a) but now encrypt "dog" as one message m.

P8. Consider RSA with $p = 5$ and $q = 11$.

a. What are n and z?

b. Let e be 3. Why is this an acceptable choice for e?

c. Find d such that $de = 1 \pmod{z}$ and $d < 160$.

d. Encrypt the message $m = 8$ using the key (n, e). Let c denote the corresponding ciphertext. Show all work. *Hint:* To simplify the calculations, use the fact:

$$[(a \bmod n) \cdot (b \bmod n)] \bmod n = (a \cdot b) \bmod n$$

P9. In this problem, we explore the Diffie-Hellman (DH) public-key encryption algorithm, which allows two entities to agree on a shared key. The DH algorithm makes use of a large prime number p and another large number g less than p. Both p and g are made public (so that an attacker would know them). In DH, Alice and Bob each independently choose secret keys, S_A and S_B, respectively. Alice then computes her public key, T_A, by raising g to S_A and then taking mod p. Bob similarly computes his own public key T_B by raising g to S_B and then taking mod p. Alice and Bob then exchange their public keys over the Internet. Alice then calculates the shared secret key S by raising T_B to S_A and then taking mod p. Similarly, Bob calculates the shared key S' by raising T_A to S_B and then taking mod p.

a. Prove that, in general, Alice and Bob obtain the same symmetric key, that is, prove $S = S'$.

b. With $p = 11$ and $g = 2$, suppose Alice and Bob choose private keys $S_A = 5$ and $S_B = 12$, respectively. Calculate Alice's and Bob's public keys, T_A and T_B. Show all work.

c. Following up on part (b), now calculate S as the shared symmetric key. Show all work.

d. Provide a timing diagram that shows how Diffie-Hellman can be attacked by a man-in-the-middle. The timing diagram should have three vertical lines, one for Alice, one for Bob, and one for the attacker Trudy.

P10. Suppose Alice wants to communicate with Bob using symmetric key cryptography using a session key K_S. In Section 8.2, we learned how public-key cryptography can be used to distribute the session key from Alice to Bob. In this problem, we explore how the session key can be distributed—without

public key cryptography—using a key distribution center (KDC). The KDC is a server that shares a unique secret symmetric key with each registered user. For Alice and Bob, denote these keys by $K_{A\text{-}KDC}$ and $K_{B\text{-}KDC}$. Design a scheme that uses the KDC to distribute K_S to Alice and Bob. Your scheme should use three messages to distribute the session key: a message from Alice to the KDC; a message from the KDC to Alice; and finally a message from Alice to Bob. The first message is $K_{A\text{-}KDC}$ (A, B). Using the notation, $K_{A\text{-}KDC}$, $K_{B\text{-}KDC}$, S, A, and B answer the following questions.

a. What is the second message?

b. What is the third message?

P11. Compute a third message, different from the two messages in Figure 8.8, that has the same checksum as the messages in Figure 8.8.

P12. Suppose Alice and Bob share two secret keys: an authentication key S_1 and a symmetric encryption key S_2. Augment Figure 8.9 so that both integrity and confidentiality are provided.

P13. In the BitTorrent P2P file distribution protocol (see Chapter 2), the seed breaks the file into blocks, and the peers redistribute the blocks to each other. Without any protection, an attacker can easily wreak havoc in a torrent by masquerading as a benevolent peer and sending bogus blocks to a small subset of peers in the torrent. These unsuspecting peers then redistribute the bogus blocks to other peers, which in turn redistribute the bogus blocks to even more peers. Thus, it is critical for BitTorrent to have a mechanism that allows a peer to verify the integrity of a block, so that it doesn't redistribute bogus blocks. Assume that when a peer joins a torrent, it initially gets a `.torrent` file from a *fully* trusted source. Describe a simple scheme that allows peers to verify the integrity of blocks.

P14. The OSPF routing protocol uses a MAC rather than digital signatures to provide message integrity. Why do you think a MAC was chosen over digital signatures?

P15. Consider our authentication protocol in Figure 8.18 in which Alice authenticates herself to Bob, which we saw works well (i.e., we found no flaws in it). Now suppose that while Alice is authenticating herself to Bob, Bob must authenticate himself to Alice. Give a scenario by which Trudy, pretending to be Alice, can now authenticate herself to Bob as Alice. (*Hint*: Consider that the sequence of operations of the protocol, one with Trudy initiating and one with Bob initiating, can be arbitrarily interleaved. Pay particular attention to the fact that both Bob and Alice will use a nonce, and that if care is not taken, the same nonce can be used maliciously.)

P16. A natural question is whether we can use a nonce and public key cryptography to solve the end-point authentication problem in Section 8.4. Consider the following natural protocol: (1) Alice sends the message "`I am Alice`" to Bob. (2) Bob chooses a nonce, R, and sends it to Alice. (3) Alice uses her *private* key to encrypt

the nonce and sends the resulting value to Bob. (4) Bob applies Alice's public key to the received message. Thus, Bob computes R and authenticates Alice.

a. Diagram this protocol, using the notation for public and private keys employed in the textbook.

b. Suppose that certificates are not used. Describe how Trudy can become a "woman-in-the-middle" by intercepting Alice's messages and then pretending to be Alice to Bob.

P17. Figure 8.19 shows the operations that Alice must perform with PGP to provide confidentiality, authentication, and integrity. Diagram the corresponding operations that Bob must perform on the package received from Alice.

P18. Suppose Alice wants to send an e-mail to Bob. Bob has a public-private key pair (K_B^+, K_B^-), and Alice has Bob's certificate. But Alice does not have a public, private key pair. Alice and Bob (and the entire world) share the same hash function $H(\cdot)$.

a. In this situation, is it possible to design a scheme so that Bob can verify that Alice created the message? If so, show how with a block diagram for Alice and Bob.

b. Is it possible to design a scheme that provides confidentiality for sending the message from Alice to Bob? If so, show how with a block diagram for Alice and Bob.

P19. Consider the Wireshark output below for a portion of an SSL session.

a. Is Wireshark packet 112 sent by the client or server?

b. What is the server's IP address and port number?

c. Assuming no loss and no retransmissions, what will be the sequence number of the next TCP segment sent by the client?

d. How many SSL records does Wireshark packet 112 contain?

e. Does packet 112 contain a Master Secret or an Encrypted Master Secret or neither?

f. Assuming that the handshake type field is 1 byte and each length field is 3 bytes, what are the values of the first and last bytes of the Master Secret (or Encrypted Master Secret)?

g. The client encrypted handshake message takes into account how many SSL records?

h. The server encrypted handshake message takes into account how many SSL records?

P20. In Section 8.6.1, it is shown that without sequence numbers, Trudy (a woman-in-the middle) can wreak havoc in an SSL session by interchanging TCP segments. Can Trudy do something similar by deleting a TCP segment? What does she need to do to succeed at the deletion attack? What effect will it have?

File Edit View Go Capture Analyze Statistics Help

Filter: ssl — Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Info
106	21.805705	128.238.38.162	216.75.194.220	SSLv2	Client Hello
108	21.830201	216.75.194.220	128.238.38.162	SSLv3	Server Hello,
111	21.853520	216.75.194.220	128.238.38.162	SSLv3	Certificate, Server Hello Done
112	21.876168	128.238.38.162	216.75.194.220	SSLv3	Client Key Exchange, Change Cipher Spec, Encrypted Handshake M
113	21.945667	216.75.194.220	128.238.38.162	SSLv3	Change Cipher Spec, Encrypted Handshake Message
114	21.954189	128.238.38.162	216.75.194.220	SSLv3	Application Data

```
⊞ Frame 112 (258 bytes on wire, 258 bytes captured)
⊞ Ethernet II, Src: Ibm_10:60:99 (00:09:6b:10:60:99), Dst: All-HSRP-routers_00 (00:00:0c:07:ac:00)
⊞ Internet Protocol, Src: 128.238.38.162 (128.238.38.162), Dst: 216.75.194.220 (216.75.194.220)
⊞ Transmission Control Protocol, Src Port: 2271 (2271), Dst Port: https (443), Seq: 79, Ack: 2785, Len: 204
⊟ Secure Socket Layer
  ⊟ SSLv3 Record Layer: Handshake Protocol: Client Key Exchange
      Content Type: Handshake (22)
      Version: SSL 3.0 (0x0300)
      Length: 132
    ⊟ Handshake Protocol: Client Key Exchange
        Handshake Type: Client Key Exchange (16)
        Length: 128
  ⊟ SSLv3 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
      Content Type: Change Cipher Spec (20)
      Version: SSL 3.0 (0x0300)
      Length: 1
      Change Cipher Spec Message
  ⊟ SSLv3 Record Layer: Handshake Protocol: Encrypted Handshake Message
      Content Type: Handshake (22)
      Version: SSL 3.0 (0x0300)
      Length: 56
      Handshake Protocol: Encrypted Handshake Message
```

```
0030  fd 1f c2 d9 00 00 16 03  00 00 84 10 00 00 80 bc   ........ ........
0040  49 49 47 29 aa 25 90 47  7f d0 59 05 6a e7 89 56   IIG).%.G ..Y.j..V
0050  c7 7b 12 af 08 b4 7c 60  9e 61 f1 04 b0 fb f8 3e   .{....|` .a.....>
0060  41 c0 8d c9 10 93 9c ad  1e ce 82 e0 dd e2 50 b9   A....... ......P.
0070  9b 4b 51 c7 3f bd ee cd  92 c4 27 5d ff dd fb 95   .KQ.?... ..']....
0080  42 3d a4 b7 71 ee c0 ff  c3 ce b2 ed 60 90 6c d7   B=..q... ....`.l.
0090  04 6e 9a 00 98 2e 52 ee  b5 bc d1 c4 f5 63 f0 e3   .nZ...R. .....C..
00a0  44 29 f1 c6 ba 64 58 79  46 9e 3e c4 fd d7 9b 7a   D)...dXy F.>....z
00b0  02 04 09 32 f6 1d 7a a1  2d cf d2 1a 18 64 29 14   ...2..z. -....d).
00c0  03 00 00 01 01 16 03 00  00 38 29 a9 dc 11 5a 74   ........ .8)...Zt
00d0  7a 41 48 15 4f 50 4b e2  df 0c d0 5b c4 44 a8 e8   zAH.OPK. ...[.D..
00e0  e4 e5 12 b9 11 f6 b3 9a  de b7 22 0d 3a 17 9a 83   ........ ..".:...
00f0  77 1c de ab f2 41 e7 2e  ad d5 1c 5b a2 0d ab e4   w....A.. ...[....
```

(Wireshark screenshot reprinted by permission of the Wireshark Foundation.)

P21. Suppose Alice and Bob are communicating over an SSL session. Suppose an attacker, who does not have any of the shared keys, inserts a bogus TCP segment into a packet stream with correct TCP checksum and sequence numbers (and correct IP addresses and port numbers). Will SSL at the receiving side accept the bogus packet and pass the payload to the receiving application? Why or why not?

P22. The following true/false questions pertain to Figure 8.28.

a. When a host in 172.16.1/24 sends a datagram to an Amazon.com server, the router R1 will encrypt the datagram using IPsec.

b. When a host in 172.16.1/24 sends a datagram to a host in 172.16.2/24, the router R1 will change the source and destination address of the IP datagram.

c. Suppose a host in 172.16.1/24 initiates a TCP connection to a Web server in 172.16.2/24. As part of this connection, all datagrams sent by R1 will have protocol number 50 in the left-most IPv4 header field.

d. Consider sending a TCP segment from a host in 172.16.1/24 to a host in 172.16.2/24. Suppose the acknowledgment for this segment gets lost, so that TCP resends the segment. Because IPsec uses sequence numbers, R1 will not resend the TCP segment.

P23. Consider the example in Figure 8.28. Suppose Trudy is a woman-in-the-middle, who can insert datagrams into the stream of datagrams going from R1 and R2. As part of a replay attack, Trudy sends a duplicate copy of one of the datagrams sent from R1 to R2. Will R2 decrypt the duplicate datagram and forward it into the branch-office network? If not, describe in detail how R2 detects the duplicate datagram.

P24. Consider the following pseudo-WEP protocol. The key is 4 bits and the IV is 2 bits. The IV is appended to the end of the key when generating the key-stream. Suppose that the shared secret key is 1010. The keystreams for the four possible inputs are as follows:

101000: 0010101101010101001011010100100 . . .

101001: 1010011011001010110100100101101 . . .

101010: 0001101000111100010100101001111 . . .

101011: 1111101010000000101010100010111 . . .

Suppose all messages are 8 bits long. Suppose the ICV (integrity check) is 4 bits long, and is calculated by XOR-ing the first 4 bits of data with the last 4 bits of data. Suppose the pseudo-WEP packet consists of three fields: first the IV field, then the message field, and last the ICV field, with some of these fields encrypted.

a. We want to send the message $m = 10100000$ using the IV $= 11$ and using WEP. What will be the values in the three WEP fields?

b. Show that when the receiver decrypts the WEP packet, it recovers the message and the ICV.

c. Suppose Trudy intercepts a WEP packet (not necessarily with the IV $= 11$) and wants to modify it before forwarding it to the receiver. Suppose Trudy flips the first ICV bit. Assuming that Trudy does not know the keystreams for any of the IVs, what other bit(s) must Trudy also flip so that the received packet passes the ICV check?

d. Justify your answer by modifying the bits in the WEP packet in part (a), decrypting the resulting packet, and verifying the integrity check.

P25. Provide a filter table and a connection table for a stateful firewall that is as restrictive as possible but accomplishes the following:

a. Allows all internal users to establish Telnet sessions with external hosts.

b. Allows external users to surf the company Web site at 222.22.0.12.

c. But otherwise blocks all inbound and outbound traffic.

The internal network is 222.22/16. In your solution, suppose that the connection table is currently caching three connections, all from inside to outside. You'll need to invent appropriate IP addresses and port numbers.

P26. Suppose Alice wants to visit the Web site activist.com using a TOR-like service. This service uses two non-colluding proxy servers, Proxy1 and Proxy2. Alice first obtains the certificates (each containing a public key) for Proxy1 and Proxy2 from some central server. Denote $K_1^+(\,)$, $K_2^+(\,)$, $K_1^-(\,)$, and $K_2^-(\,)$ for the encryption/decryption with public and private RSA keys.

a. Using a timing diagram, provide a protocol (as simple as possible) that enables Alice to establish a shared session key S_1 with Proxy1. Denote $S_1(m)$ for encryption/decryption of data m with the shared key S_1.

b. Using a timing diagram, provide a protocol (as simple as possible) that allows Alice to establish a shared session key S_2 with Proxy2 *without revealing her IP address to Proxy2*.

c. Assume now that shared keys S_1 and S_2 are now established. Using a timing diagram, provide a protocol (as simple as possible and *not using public-key cryptography*) that allows Alice to request an html page from activist.com *without revealing her IP address to Proxy2* and *without revealing to Proxy1 which site she is visiting*. Your diagram should end with an HTTP request arriving at activist.com.

Wireshark Lab

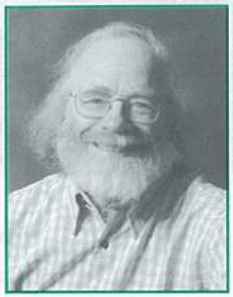
In this lab (available from the book Web site), we investigate the Secure Sockets Layer (SSL) protocol. Recall from Section 8.6 that SSL is used for securing a TCP connection, and that it is extensively used in practice for secure Internet transactions. In this lab, we will focus on the SSL records sent over the TCP connection. We will attempt to delineate and classify each of the records, with a goal of understanding the why and how for each record. We investigate the various SSL record types as well as the fields in the SSL messages. We do so by analyzing a trace of the SSL records sent between your host and an e-commerce server.

IPsec Lab

In this lab (available from the book Web site), we will explore how to create IPsec SAs between linux boxes. You can do the first part of the lab with two ordinary linux boxes, each with one Ethernet adapter. But for the second part of the lab, you will need four linux boxes, two of which having two Ethernet adapters. In the second half of the lab, you will create IPsec SAs using the ESP protocol in the tunnel mode. You will do this by first manually creating the SAs, and then by having IKE create the SAs.

AN INTERVIEW WITH...

Steven M. Bellovin



Steven M. Bellovin joined the faculty at Columbia University after many years at the Network Services Research Lab at AT&T Labs Research in Florham Park, New Jersey. His focus is on networks, security, and why the two are incompatible. In 1995, he was awarded the Usenix Lifetime Achievement Award for his work in the creation of Usenet, the first newsgroup exchange network that linked two or more computers and allowed users to share information and join in discussions. Steve is also an elected member of the National Academy of Engineering. He received his BA from Columbia University and his PhD from the University of North Carolina at Chapel Hill.

What led you to specialize in the networking security area?

This is going to sound odd, but the answer is simple: It was fun. My background was in systems programming and systems administration, which leads fairly naturally to security. And I've always been interested in communications, ranging back to part-time systems programming jobs when I was in college.

My work on security continues to be motivated by two things—a desire to keep computers useful, which means that their function can't be corrupted by attackers, and a desire to protect privacy.

What was your vision for Usenet at the time that you were developing it? And now?

We originally viewed it as a way to talk about computer science and computer programming around the country, with a lot of local use for administrative matters, for-sale ads, and so on. In fact, my original prediction was one to two messages per day, from 50–100 sites at the most—ever. But the real growth was in people-related topics, including—but not limited to—human interactions with computers. My favorite newsgroups, over the years, have been things like rec.woodworking, as well as sci.crypt.

To some extent, netnews has been displaced by the Web. Were I to start designing it today, it would look very different. But it still excels as a way to reach a very broad audience that is interested in the topic, without having to rely on particular Web sites.

Has anyone inspired you professionally? In what ways?

Professor Fred Brooks—the founder and original chair of the computer science department at the University of North Carolina at Chapel Hill, the manager of the team that developed the IBM S/360 and OS/360, and the author of *The Mythical Man-Month*—was a tremendous

influence on my career. More than anything else, he taught outlook and trade-offs—how to look at problems in the context of the real world (and how much messier the real world is than a theorist would like), and how to balance competing interests in designing a solution. Most computer work is engineering—the art of making the right trade-offs to satisfy many contradictory objectives.

What is your vision for the future of networking and security?

Thus far, much of the security we have has come from isolation. A firewall, for example, works by cutting off access to certain machines and services. But we're in an era of increasing connectivity—it's gotten harder to isolate things. Worse yet, our production systems require far more separate pieces, interconnected by networks. Securing all that is one of our biggest challenges.

What would you say have been the greatest advances in security? How much further do we have to go?

At least scientifically, we know how to do cryptography. That's been a big help. But most security problems are due to buggy code, and that's a much harder problem. In fact, it's the oldest unsolved problem in computer science, and I think it will remain that way. The challenge is figuring out how to secure systems when we have to build them out of insecure components. We can already do that for reliability in the face of hardware failures; can we do the same for security?

Do you have any advice for students about the Internet and networking security?

Learning the mechanisms is the easy part. Learning how to "think paranoid" is harder. You have to remember that probability distributions don't apply—the attackers can and will find improbable conditions. And the details matter—a lot.

CHAPTER 9

Multimedia Networking

While lounging in bed or riding buses and subways, people in all corners of the world are currently using the Internet to watch movies and television shows on demand. Internet movie and television distribution companies such as Netflix and Amazon in North America and Youku and Kankan in China have practically become household names. But people are not only watching Internet videos, they are using sites like YouTube to upload and distribute their own user-generated content, becoming Internet video producers as well as consumers. Moreover, network applications such as Skype, Google Talk, and WeChat (enormously popular in China) allow people to not only make "telephone calls" over the Internet, but to also enhance those calls with video and multi-person conferencing. In fact, we predict that by the end of the current decade most of the video consumption and voice conversations will take place end-to-end over the Internet, more typically to wireless devices connected to the Internet via cellular and WiFi access networks. Traditional telephony and broadcast television are quickly becoming obsolete.

We begin this chapter with a taxonomy of multimedia applications in Section 9.1. We'll see that a multimedia application can be classified as either *streaming stored audio/video*, *conversational voice/video-over-IP*, or *streaming live audio/video*. We'll see that each of these classes of applications has its own unique service requirements that differ significantly from those of traditional elastic applications such as e-mail, Web browsing, and remote login. In Section 9.2, we'll examine video streaming in some detail. We'll explore many of the underlying principles behind

video streaming, including client buffering, prefetching, and adapting video quality to available bandwidth. In Section 9.3, we investigate conversational voice and video, which, unlike elastic applications, are highly sensitive to end-to-end delay but can tolerate occasional loss of data. Here we'll examine how techniques such as adaptive playout, forward error correction, and error concealment can mitigate against network-induced packet loss and delay. We'll also examine Skype as a case study. In Section 9.4, we'll study RTP and SIP, two popular protocols for real-time conversational voice and video applications. In Section 9.5, we'll investigate mechanisms within the network that can be used to distinguish one class of traffic (e.g., delay-sensitive applications such as conversational voice) from another (e.g., elastic applications such as browsing Web pages), and provide differentiated service among multiple classes of traffic.

9.1 Multimedia Networking Applications

We define a multimedia network application as any network application that employs audio or video. In this section, we provide a taxonomy of multimedia applications. We'll see that each class of applications in the taxonomy has its own unique set of service requirements and design issues. But before diving into an in-depth discussion of Internet multimedia applications, it is useful to consider the intrinsic characteristics of the audio and video media themselves.

9.1.1 Properties of Video

Perhaps the most salient characteristic of video is its **high bit rate**. Video distributed over the Internet typically ranges from 100 kbps for low-quality video conferencing to over 3 Mbps for streaming high-definition movies. To get a sense of how video bandwidth demands compare with those of other Internet applications, let's briefly consider three different users, each using a different Internet application. Our first user, Frank, is going quickly through photos posted on his friends' Facebook pages. Let's assume that Frank is looking at a new photo every 10 seconds, and that photos are on average 200 Kbytes in size. (As usual, throughout this discussion we make the simplifying assumption that 1 Kbyte = 8,000 bits.) Our second user, Martha, is streaming music from the Internet ("the cloud") to her smartphone. Let's assume Martha is using a service such as Spotify to listen to many MP3 songs, one after the other, each encoded at a rate of 128 kbps. Our third user, Victor, is watching a video that has been encoded at 2 Mbps. Finally, let's suppose that the session length for all three users is 4,000 seconds (approximately 67 minutes). Table 9.1 compares the bit rates and the total bytes transferred for these three users. We see that video streaming consumes by far the most bandwidth, having a bit rate of more than ten times greater than that of the Facebook and music-streaming applications. Therefore, when design-

	Bit rate	Bytes transferred in 67 min
Facebook Frank	160 kbps	80 Mbytes
Martha Music	128 kbps	64 Mbytes
Victor Video	2 Mbps	1 Gbyte

Table 9.1 ♦ Comparison of bit-rate requirements of three Internet applications

ing networked video applications, the first thing we must keep in mind is the high bit-rate requirements of video. Given the popularity of video and its high bit rate, it is perhaps not surprising that Cisco predicts [Cisco 2015] that streaming and stored video will be approximately 80 percent of global consumer Internet traffic by 2019.

Another important characteristic of video is that it can be compressed, thereby trading off video quality with bit rate. A video is a sequence of images, typically being displayed at a constant rate, for example, at 24 or 30 images per second. An uncompressed, digitally encoded image consists of an array of pixels, with each pixel encoded into a number of bits to represent luminance and color. There are two types of redundancy in video, both of which can be exploited by **video compression**. *Spatial redundancy* is the redundancy within a given image. Intuitively, an image that consists of mostly white space has a high degree of redundancy and can be efficiently compressed without significantly sacrificing image quality. *Temporal redundancy* reflects repetition from image to subsequent image. If, for example, an image and the subsequent image are exactly the same, there is no reason to re-encode the subsequent image; it is instead more efficient simply to indicate during encoding that the subsequent image is exactly the same. Today's off-the-shelf compression algorithms can compress a video to essentially any bit rate desired. Of course, the higher the bit rate, the better the image quality and the better the overall user viewing experience.

We can also use compression to create **multiple versions** of the same video, each at a different quality level. For example, we can use compression to create, say, three versions of the same video, at rates of 300 kbps, 1 Mbps, and 3 Mbps. Users can then decide which version they want to watch as a function of their current available bandwidth. Users with high-speed Internet connections might choose the 3 Mbps version; users watching the video over 3G with a smartphone might choose the 300 kbps version. Similarly, the video in a video conference application can be compressed "on-the-fly" to provide the best video quality given the available end-to-end bandwidth between conversing users.

9.1.2 Properties of Audio

Digital audio (including digitized speech and music) has significantly lower bandwidth requirements than video. Digital audio, however, has its own unique properties that must be considered when designing multimedia network applications.

To understand these properties, let's first consider how analog audio (which humans and musical instruments generate) is converted to a digital signal:

- The analog audio signal is sampled at some fixed rate, for example, at 8,000 samples per second. The value of each sample will be some real number.
- Each of the samples is then rounded to one of a finite number of values. This operation is referred to as **quantization**. The number of such finite values—called quantization values—is typically a power of two, for example, 256 quantization values.
- Each of the quantization values is represented by a fixed number of bits. For example, if there are 256 quantization values, then each value—and hence each audio sample—is represented by one byte. The bit representations of all the samples are then concatenated together to form the digital representation of the signal. As an example, if an analog audio signal is sampled at 8,000 samples per second and each sample is quantized and represented by 8 bits, then the resulting digital signal will have a rate of 64,000 bits per second. For playback through audio speakers, the digital signal can then be converted back—that is, decoded—to an analog signal. However, the decoded analog signal is only an approximation of the original signal, and the sound quality may be noticeably degraded (for example, high-frequency sounds may be missing in the decoded signal). By increasing the sampling rate and the number of quantization values, the decoded signal can better approximate the original analog signal. Thus (as with video), there is a trade-off between the quality of the decoded signal and the bit-rate and storage requirements of the digital signal.

The basic encoding technique that we just described is called **pulse code modulation (PCM)**. Speech encoding often uses PCM, with a sampling rate of 8,000 samples per second and 8 bits per sample, resulting in a rate of 64 kbps. The audio compact disk (CD) also uses PCM, with a sampling rate of 44,100 samples per second with 16 bits per sample; this gives a rate of 705.6 kbps for mono and 1.411 Mbps for stereo.

PCM-encoded speech and music, however, are rarely used in the Internet. Instead, as with video, compression techniques are used to reduce the bit rates of the stream. Human speech can be compressed to less than 10 kbps and still be intelligible. A popular compression technique for near CD-quality stereo music is **MPEG 1 layer 3**, more commonly known as **MP3**. MP3 encoders can compress to many different rates; 128 kbps is the most common encoding rate and produces very little sound degradation. A related standard is **Advanced Audio Coding (AAC)**, which has been popularized by Apple. As with video, multiple versions of a prerecorded audio stream can be created, each at a different bit rate.

Although audio bit rates are generally much less than those of video, users are generally much more sensitive to audio glitches than video glitches. Consider, for example, a video conference taking place over the Internet. If, from time to time, the video signal is lost for a few seconds, the video conference can likely proceed

without too much user frustration. If, however, the audio signal is frequently lost, the users may have to terminate the session.

9.1.3 Types of Multimedia Network Applications

The Internet supports a large variety of useful and entertaining multimedia applications. In this subsection, we classify multimedia applications into three broad categories: *(i) streaming stored audio/video*, *(ii) conversational voice/video-over-IP*, and *(iii) streaming live audio/video*. As we will soon see, each of these application categories has its own set of service requirements and design issues.

Streaming Stored Audio and Video

To keep the discussion concrete, we focus here on streaming stored video, which typically combines video and audio components. Streaming stored audio (such as Spotify's streaming music service) is very similar to streaming stored video, although the bit rates are typically much lower.

In this class of applications, the underlying medium is prerecorded video, such as a movie, a television show, a prerecorded sporting event, or a prerecorded user-generated video (such as those commonly seen on YouTube). These prerecorded videos are placed on servers, and users send requests to the servers to view the videos *on demand*. Many Internet companies today provide streaming video, including YouTube (Google), Netflix, Amazon, and Hulu. Streaming stored video has three key distinguishing features.

- *Streaming.* In a streaming stored video application, the client typically begins video playout within a few seconds after it begins receiving the video from the server. This means that the client will be playing out from one location in the video while at the same time receiving later parts of the video from the server. This technique, known as **streaming**, avoids having to download the entire video file (and incurring a potentially long delay) before playout begins.
- *Interactivity.* Because the media is prerecorded, the user may pause, reposition forward, reposition backward, fast-forward, and so on through the video content. The time from when the user makes such a request until the action manifests itself at the client should be less than a few seconds for acceptable responsiveness.
- *Continuous playout.* Once playout of the video begins, it should proceed according to the original timing of the recording. Therefore, data must be received from the server in time for its playout at the client; otherwise, users experience video frame freezing (when the client waits for the delayed frames) or frame skipping (when the client skips over delayed frames).

By far, the most important performance measure for streaming video is average throughput. In order to provide continuous playout, the network must provide an

average throughput to the streaming application that is at least as large the bit rate of the video itself. As we will see in Section 9.2, by using buffering and prefetching, it is possible to provide continuous playout even when the throughput fluctuates, as long as the average throughput (averaged over 5–10 seconds) remains above the video rate [Wang 2008].

For many streaming video applications, prerecorded video is stored on, and streamed from, a CDN rather than from a single data center. There are also many P2P video streaming applications for which the video is stored on users' hosts (peers), with different chunks of video arriving from different peers that may spread around the globe. Given the prominence of Internet video streaming, we will explore video streaming in some depth in Section 9.2, paying particular attention to client buffering, prefetching, adapting quality to bandwidth availability, and CDN distribution.

Conversational Voice- and Video-over-IP

Real-time conversational voice over the Internet is often referred to as **Internet telephony**, since, from the user's perspective, it is similar to the traditional circuit-switched telephone service. It is also commonly called **Voice-over-IP (VoIP)**. Conversational video is similar, except that it includes the video of the participants as well as their voices. Most of today's voice and video conversational systems allow users to create conferences with three or more participants. Conversational voice and video are widely used in the Internet today, with the Internet companies Skype, QQ, and Google Talk boasting hundreds of millions of daily users.

In our discussion of application service requirements in Chapter 2 (Figure 2.4), we identified a number of axes along which application requirements can be classified. Two of these axes—timing considerations and tolerance of data loss—are particularly important for conversational voice and video applications. Timing considerations are important because audio and video conversational applications are highly **delay-sensitive**. For a conversation with two or more interacting speakers, the delay from when a user speaks or moves until the action is manifested at the other end should be less than a few hundred milliseconds. For voice, delays smaller than 150 milliseconds are not perceived by a human listener, delays between 150 and 400 milliseconds can be acceptable, and delays exceeding 400 milliseconds can result in frustrating, if not completely unintelligible, voice conversations.

On the other hand, conversational multimedia applications are **loss-tolerant**—occasional loss only causes occasional glitches in audio/video playback, and these losses can often be partially or fully concealed. These delay-sensitive but loss-tolerant characteristics are clearly different from those of elastic data applications such as Web browsing, e-mail, social networks, and remote login. For elastic applications, long delays are annoying but not particularly harmful; the completeness and integrity of the transferred data, however, are of paramount importance. We will explore conversational voice and video in more depth in Section 9.3, paying particular attention

to how adaptive playout, forward error correction, and error concealment can mitigate against network-induced packet loss and delay.

Streaming Live Audio and Video

This third class of applications is similar to traditional broadcast radio and television, except that transmission takes place over the Internet. These applications allow a user to receive a *live* radio or television transmission—such as a live sporting event or an ongoing news event—transmitted from any corner of the world. Today, thousands of radio and television stations around the world are broadcasting content over the Internet.

Live, broadcast-like applications often have many users who receive the same audio/video program at the same time. In the Internet today, this is typically done with CDNs (Section 2.6). As with streaming stored multimedia, the network must provide each live multimedia flow with an average throughput that is larger than the video consumption rate. Because the event is live, delay can also be an issue, although the timing constraints are much less stringent than those for conversational voice. Delays of up to ten seconds or so from when the user chooses to view a live transmission to when playout begins can be tolerated. We will not cover streaming live media in this book because many of the techniques used for streaming live media—initial buffering delay, adaptive bandwidth use, and CDN distribution—are similar to those for streaming stored media.

9.2 Streaming Stored Video

For streaming video applications, prerecorded videos are placed on servers, and users send requests to these servers to view the videos on demand. The user may watch the video from beginning to end without interruption, may stop watching the video well before it ends, or interact with the video by pausing or repositioning to a future or past scene. Streaming video systems can be classified into three categories: **UDP streaming**, **HTTP streaming**, and **adaptive HTTP streaming** (see Section 2.6). Although all three types of systems are used in practice, the majority of today's systems employ HTTP streaming and adaptive HTTP streaming.

A common characteristic of all three forms of video streaming is the extensive use of client-side application buffering to mitigate the effects of varying end-to-end delays and varying amounts of available bandwidth between server and client. For streaming video (both stored and live), users generally can tolerate a small several-second initial delay between when the client requests a video and when video playout begins at the client. Consequently, when the video starts to arrive at the client, the client need not immediately begin playout, but can instead build up a reserve of video in an application buffer. Once the client has built up a reserve of several seconds of

buffered-but-not-yet-played video, the client can then begin video playout. There are two important advantages provided by such **client buffering**. First, client-side buffering can absorb variations in server-to-client delay. If a particular piece of video data is delayed, as long as it arrives before the reserve of received-but-not-yet-played video is exhausted, this long delay will not be noticed. Second, if the server-to-client bandwidth briefly drops below the video consumption rate, a user can continue to enjoy continuous playback, again as long as the client application buffer does not become completely drained.

Figure 9.1 illustrates client-side buffering. In this simple example, suppose that video is encoded at a fixed bit rate, and thus each video block contains video frames that are to be played out over the same fixed amount of time, Δ. The server transmits the first video block at t_0, the second block at $t_0 + \Delta$, the third block at $t_0 + 2\Delta$, and so on. Once the client begins playout, each block should be played out Δ time units after the previous block in order to reproduce the timing of the original recorded video. Because of the variable end-to-end network delays, different video blocks experience different delays. The first video block arrives at the client at t_1 and the second block arrives at t_2. The network delay for the ith block is the horizontal distance between the time the block was transmitted by the server and the time it is received at the client; note that the network delay varies from one video block to another. In this example, if the client were to begin playout as soon as the first block arrived at t_1, then the second block would not have arrived in time to be played out at out at $t_1 + \Delta$. In this case, video playout would either have to stall (waiting for block 2 to arrive) or block 2 could be skipped—both resulting in undesirable playout impairments. Instead, if the client were to delay the start of playout until t_3, when blocks 1 through 6 have all arrived, periodic playout can proceed with *all* blocks having been received before their playout time.

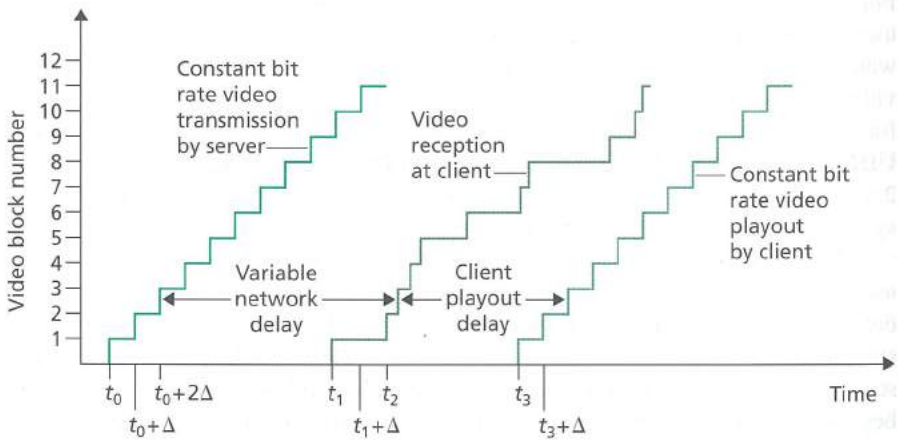


Figure 9.1 ♦ Client playout delay in video streaming

9.2.1 UDP Streaming

We only briefly discuss UDP streaming here, referring the reader to more in-depth discussions of the protocols behind these systems where appropriate. With UDP streaming, the server transmits video at a rate that matches the client's video consumption rate by clocking out the video chunks over UDP at a steady rate. For example, if the video consumption rate is 2 Mbps and each UDP packet carries 8,000 bits of video, then the server would transmit one UDP packet into its socket every (8000 bits)/(2 Mbps) = 4 msec. As we learned in Chapter 3, because UDP does not employ a congestion-control mechanism, the server can push packets into the network at the consumption rate of the video without the rate-control restrictions of TCP. UDP streaming typically uses a small client-side buffer, big enough to hold less than a second of video.

Before passing the video chunks to UDP, the server will encapsulate the video chunks within transport packets specially designed for transporting audio and video, using the Real-Time Transport Protocol (RTP) [RFC 3550] or a similar (possibly proprietary) scheme. We delay our coverage of RTP until Section 9.3, where we discuss RTP in the context of conversational voice and video systems.

Another distinguishing property of UDP streaming is that in addition to the server-to-client video stream, the client and server also maintain, in parallel, a separate control connection over which the client sends commands regarding session state changes (such as pause, resume, reposition, and so on). The Real-Time Streaming Protocol (RTSP) [RFC 2326], explained in some detail in the Web site for this textbook, is a popular open protocol for such a control connection.

Although UDP streaming has been employed in many open-source systems and proprietary products, it suffers from three significant drawbacks. First, due to the unpredictable and varying amount of available bandwidth between server and client, constant-rate UDP streaming can fail to provide continuous playout. For example, consider the scenario where the video consumption rate is 1 Mbps and the server-to-client available bandwidth is usually more than 1 Mbps, but every few minutes the available bandwidth drops below 1 Mbps for several seconds. In such a scenario, a UDP streaming system that transmits video at a constant rate of 1 Mbps over RTP/UDP would likely provide a poor user experience, with freezing or skipped frames soon after the available bandwidth falls below 1 Mbps. The second drawback of UDP streaming is that it requires a media control server, such as an RTSP server, to process client-to-server interactivity requests and to track client state (e.g., the client's playout point in the video, whether the video is being paused or played, and so on) for *each* ongoing client session. This increases the overall cost and complexity of deploying a large-scale video-on-demand system. The third drawback is that many firewalls are configured to block UDP traffic, preventing the users behind these firewalls from receiving UDP video.

9.2.2 HTTP Streaming

In HTTP streaming, the video is simply stored in an HTTP server as an ordinary file with a specific URL. When a user wants to see the video, the client establishes a TCP connection with the server and issues an HTTP GET request for that URL. The server then sends the video file, within an HTTP response message, as quickly as possible, that is, as quickly as TCP congestion control and flow control will allow. On the client side, the bytes are collected in a client application buffer. Once the number of bytes in this buffer exceeds a predetermined threshold, the client application begins playback—specifically, it periodically grabs video frames from the client application buffer, decompresses the frames, and displays them on the user's screen.

We learned in Chapter 3 that when transferring a file over TCP, the server-to-client transmission rate can vary significantly due to TCP's congestion control mechanism. In particular, it is not uncommon for the transmission rate to vary in a "saw-tooth" manner associated with TCP congestion control. Furthermore, packets can also be significantly delayed due to TCP's retransmission mechanism. Because of these characteristics of TCP, the conventional wisdom in the 1990s was that video streaming would never work well over TCP. Over time, however, designers of streaming video systems learned that TCP's congestion control and reliable-data transfer mechanisms do not necessarily preclude continuous playout when client buffering and prefetching (discussed in the next section) are used.

The use of HTTP over TCP also allows the video to traverse firewalls and NATs more easily (which are often configured to block most UDP traffic but to allow most HTTP traffic). Streaming over HTTP also obviates the need for a media control server, such as an RTSP server, reducing the cost of a large-scale deployment over the Internet. Due to all of these advantages, most video streaming applications today—including YouTube and Netflix—use HTTP streaming (over TCP) as its underlying streaming protocol.

Prefetching Video

As we just learned, client-side buffering can be used to mitigate the effects of varying end-to-end delays and varying available bandwidth. In our earlier example in Figure 9.1, the server transmits video at the rate at which the video is to be played out. However, for streaming *stored* video, the client can attempt to download the video at a rate *higher* than the consumption rate, thereby **prefetching** video frames that are to be consumed in the future. This prefetched video is naturally stored in the client application buffer. Such prefetching occurs naturally with TCP streaming, since TCP's congestion avoidance mechanism will attempt to use all of the available bandwidth between server and client.

To gain some insight into prefetching, let's take a look at a simple example. Suppose the video consumption rate is 1 Mbps but the network is capable of delivering the video from server to client at a constant rate of 1.5 Mbps. Then the client will

not only be able to play out the video with a very small playout delay, but will also be able to increase the amount of buffered video data by 500 Kbits every second. In this manner, if in the future the client receives data at a rate of less than 1 Mbps for a brief period of time, the client will be able to continue to provide continuous playback due to the reserve in its buffer. [Wang 2008] shows that when the average TCP throughput is roughly twice the media bit rate, streaming over TCP results in minimal starvation and low buffering delays.

Client Application Buffer and TCP Buffers

Figure 9.2 illustrates the interaction between client and server for HTTP streaming. At the server side, the portion of the video file in white has already been sent into the server's socket, while the darkened portion is what remains to be sent. After "passing through the socket door," the bytes are placed in the TCP send buffer before being transmitted into the Internet, as described in Chapter 3. In Figure 9.2, because the TCP send buffer at the server side is shown to be full, the server is momentarily prevented from sending more bytes from the video file into the socket. On the client side, the client application (media player) reads bytes from the TCP receive buffer (through its client socket) and places the bytes into the client application buffer. At the same time, the client application periodically grabs video frames from the client application buffer, decompresses the frames, and displays them on the user's screen. Note that if the client application buffer is larger than the video file, then the whole process of moving bytes from the server's storage to the client's application buffer is equivalent to an ordinary file download over HTTP—the client simply pulls the video off the server as fast as TCP will allow!

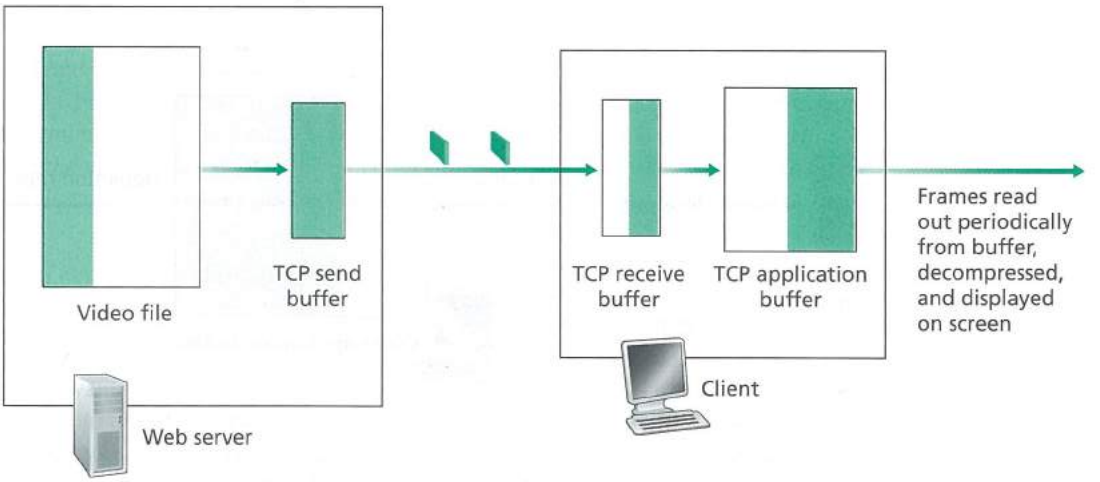


Figure 9.2 ♦ Streaming stored video over HTTP/TCP

Consider now what happens when the user pauses the video during the streaming process. During the pause period, bits are not removed from the client application buffer, even though bits continue to enter the buffer from the server. If the client application buffer is finite, it may eventually become full, which will cause "back pressure" all the way back to the server. Specifically, once the client application buffer becomes full, bytes can no longer be removed from the client TCP receive buffer, so it too becomes full. Once the client receive TCP buffer becomes full, bytes can no longer be removed from the server TCP send buffer, so it also becomes full. Once the TCP becomes full, the server cannot send any more bytes into the socket. Thus, if the user pauses the video, the server may be forced to stop transmitting, in which case the server will be blocked until the user resumes the video.

In fact, even during regular playback (that is, without pausing), if the client application buffer becomes full, back pressure will cause the TCP buffers to become full, which will force the server to reduce its rate. To determine the resulting rate, note that when the client application removes f bits, it creates room for f bits in the client application buffer, which in turn allows the server to send f additional bits. Thus, the server send rate can be no higher than the video consumption rate at the client. Therefore, *a full client application buffer indirectly imposes a limit on the rate that video can be sent from server to client when streaming over HTTP.*

Analysis of Video Streaming

Some simple modeling will provide more insight into initial playout delay and freezing due to application buffer depletion. As shown in Figure 9.3, let B denote the size

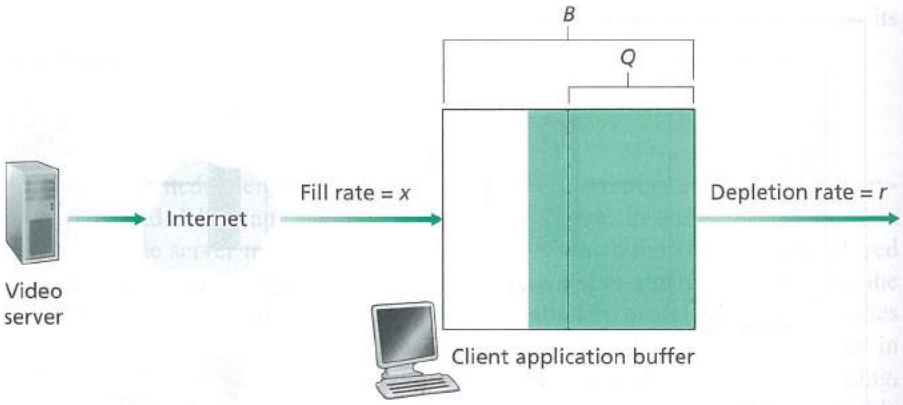


Figure 9.3 ♦ Analysis of client-side buffering for video streaming

(in bits) of the client's application buffer, and let Q denote the number of bits that must be buffered before the client application begins playout. (Of course, $Q < B$.) Let r denote the video consumption rate—the rate at which the client draws bits out of the client application buffer during playback. So, for example, if the video's frame rate is 30 frames/sec, and each (compressed) frame is 100,000 bits, then $r = 3$ Mbps. To see the forest through the trees, we'll ignore TCP's send and receive buffers.

Let's assume that the server sends bits at a constant rate x whenever the client buffer is not full. (This is a gross simplification, since TCP's send rate varies due to congestion control; we'll examine more realistic time-dependent rates $x(t)$ in the problems at the end of this chapter.) Suppose at time $t = 0$, the application buffer is empty and video begins arriving to the client application buffer. We now ask at what time $t = t_p$ does playout begin? And while we are at it, at what time $t = t_f$ does the client application buffer become full?

First, let's determine t_p, the time when Q bits have entered the application buffer and playout begins. Recall that bits arrive to the client application buffer at rate x and *no* bits are removed from this buffer before playout begins. Thus, the amount of time required to build up Q bits (the initial buffering delay) is $t_p = Q/x$.

Now let's determine t_f, the point in time when the client application buffer becomes full. We first observe that if $x < r$ (that is, if the server send rate is less than the video consumption rate), then the client buffer will never become full! Indeed, starting at time t_p, the buffer will be depleted at rate r and will only be filled at rate $x < r$. Eventually the client buffer will empty out entirely, at which time the video will freeze on the screen while the client buffer waits another t_p seconds to build up Q bits of video. *Thus, when the available rate in the network is less than the video rate, playout will alternate between periods of continuous playout and periods of freezing.* In a homework problem, you will be asked to determine the length of each continuous playout and freezing period as a function of Q, r, and x. Now let's determine t_f for when $x > r$. In this case, starting at time t_p, the buffer increases from Q to B at rate $x - r$ since bits are being depleted at rate r but are arriving at rate x, as shown in Figure 9.3. Given these hints, you will be asked in a homework problem to determine t_f, the time the client buffer becomes full. Note that *when the available rate in the network is more than the video rate, after the initial buffering delay, the user will enjoy continuous playout until the video ends.*

Early Termination and Repositioning the Video

HTTP streaming systems often make use of the **HTTP byte-range header** in the HTTP GET request message, which specifies the specific range of bytes the client currently wants to retrieve from the desired video. This is particularly useful when the user wants to reposition (that is, jump) to a future point in time in the video. When the user repositions to a new position, the client sends a new HTTP request, indicating with the byte-range header from which byte in the file should the server send data. When the server receives the new HTTP request, it can forget about any earlier request and instead send bytes beginning with the byte indicated in the byte-range request.

While we are on the subject of repositioning, we briefly mention that when a user repositions to a future point in the video or terminates the video early, some prefetched-but-not-yet-viewed data transmitted by the server will go unwatched—a waste of network bandwidth and server resources. For example, suppose that the client buffer is full with B bits at some time t_0 into the video, and at this time the user repositions to some instant $t > t_0 + B/r$ into the video, and then watches the video to completion from that point on. In this case, all B bits in the buffer will be unwatched and the bandwidth and server resources that were used to transmit those B bits have been completely wasted. There is significant wasted bandwidth in the Internet due to early termination, which can be quite costly, particularly for wireless links [Ihm 2011]. For this reason, many streaming systems use only a moderate-size client application buffer, or will limit the amount of prefetched video using the byte-range header in HTTP requests [Rao 2011].

Repositioning and early termination are analogous to cooking a large meal, eating only a portion of it, and throwing the rest away, thereby wasting food. So the next time your parents criticize you for wasting food by not eating all your dinner, you can quickly retort by saying they are wasting bandwidth and server resources when they reposition while watching movies over the Internet! But, of course, two wrongs do not make a right—both food and bandwidth are not to be wasted!

In Sections 9.2.1 and 9.2.2, we covered UDP streaming and HTTP streaming, respectively. A third type of streaming is Dynamic Adaptive Streaming over HTTP (DASH), which uses multiple versions of the video, each compressed at a different rate. DASH is discussed in detail in Section 2.6.2. CDNs are often used to distribute stored and live video. CDNs are discussed in detail in Section 2.6.3.

9.3 Voice-over-IP

Real-time conversational voice over the Internet is often referred to as **Internet telephony**, since, from the user's perspective, it is similar to the traditional circuit-switched telephone service. It is also commonly called **Voice-over-IP (VoIP)**. In this section we describe the principles and protocols underlying VoIP. Conversational video is similar in many respects to VoIP, except that it includes the video of the participants as well as their voices. To keep the discussion focused and concrete, we focus here only on voice in this section rather than combined voice and video.

9.3.1 Limitations of the Best-Effort IP Service

The Internet's network-layer protocol, IP, provides best-effort service. That is to say the service makes its best effort to move each datagram from source to destination as quickly as possible but makes no promises whatsoever about getting the packet

to the destination within some delay bound or about a limit on the percentage of packets lost. The lack of such guarantees poses significant challenges to the design of real-time conversational applications, which are acutely sensitive to packet delay, jitter, and loss.

In this section, we'll cover several ways in which the performance of VoIP over a best-effort network can be enhanced. Our focus will be on application-layer techniques, that is, approaches that do not require any changes in the network core or even in the transport layer at the end hosts. To keep the discussion concrete, we'll discuss the limitations of best-effort IP service in the context of a specific VoIP example. The sender generates bytes at a rate of 8,000 bytes per second; every 20 msecs the sender gathers these bytes into a chunk. A chunk and a special header (discussed below) are encapsulated in a UDP segment, via a call to the socket interface. Thus, the number of bytes in a chunk is (20 msecs) · (8,000 bytes/sec) = 160 bytes, and a UDP segment is sent every 20 msecs.

If each packet makes it to the receiver with a constant end-to-end delay, then packets arrive at the receiver periodically every 20 msecs. In these ideal conditions, the receiver can simply play back each chunk as soon as it arrives. But unfortunately, some packets can be lost and most packets will not have the same end-to-end delay, even in a lightly congested Internet. For this reason, the receiver must take more care in determining (1) when to play back a chunk, and (2) what to do with a missing chunk.

Packet Loss

Consider one of the UDP segments generated by our VoIP application. The UDP segment is encapsulated in an IP datagram. As the datagram wanders through the network, it passes through router buffers (that is, queues) while waiting for transmission on outbound links. It is possible that one or more of the buffers in the path from sender to receiver is full, in which case the arriving IP datagram may be discarded, never to arrive at the receiving application.

Loss could be eliminated by sending the packets over TCP (which provides for reliable data transfer) rather than over UDP. However, retransmission mechanisms are often considered unacceptable for conversational real-time audio applications such as VoIP, because they increase end-to-end delay [Bolot 1996]. Furthermore, due to TCP congestion control, packet loss may result in a reduction of the TCP sender's transmission rate to a rate that is lower than the receiver's drain rate, possibly leading to buffer starvation. This can have a severe impact on voice intelligibility at the receiver. For these reasons, most existing VoIP applications run over UDP by default. [Baset 2006] reports that UDP is used by Skype unless a user is behind a NAT or firewall that blocks UDP segments (in which case TCP is used).

But losing packets is not necessarily as disastrous as one might think. Indeed, packet loss rates between 1 and 20 percent can be tolerated, depending on how voice is encoded and transmitted, and on how the loss is concealed at the receiver. For example, forward error correction (FEC) can help conceal packet loss. We'll see

below that with FEC, redundant information is transmitted along with the original information so that some of the lost original data can be recovered from the redundant information. Nevertheless, if one or more of the links between sender and receiver is severely congested, and packet loss exceeds 10 to 20 percent (for example, on a wireless link), then there is really nothing that can be done to achieve acceptable audio quality. Clearly, best-effort service has its limitations.

End-to-End Delay

End-to-end delay is the accumulation of transmission, processing, and queuing delays in routers; propagation delays in links; and end-system processing delays. For real-time conversational applications, such as VoIP, end-to-end delays smaller than 150 msecs are not perceived by a human listener; delays between 150 and 400 msecs can be acceptable but are not ideal; and delays exceeding 400 msecs can seriously hinder the interactivity in voice conversations. The receiving side of a VoIP application will typically disregard any packets that are delayed more than a certain threshold, for example, more than 400 msecs. Thus, packets that are delayed by more than the threshold are effectively lost.

Packet Jitter

A crucial component of end-to-end delay is the varying queuing delays that a packet experiences in the network's routers. Because of these varying delays, the time from when a packet is generated at the source until it is received at the receiver can fluctuate from packet to packet, as shown in Figure 9.1. This phenomenon is called **jitter**. As an example, consider two consecutive packets in our VoIP application. The sender sends the second packet 20 msecs after sending the first packet. But at the receiver, the spacing between these packets can become greater than 20 msecs. To see this, suppose the first packet arrives at a nearly empty queue at a router, but just before the second packet arrives at the queue a large number of packets from other sources arrive at the same queue. Because the first packet experiences a small queuing delay and the second packet suffers a large queuing delay at this router, the first and second packets become spaced by more than 20 msecs. The spacing between consecutive packets can also become less than 20 msecs. To see this, again consider two consecutive packets. Suppose the first packet joins the end of a queue with a large number of packets, and the second packet arrives at the queue before this first packet is transmitted and before any packets from other sources arrive at the queue. In this case, our two packets find themselves one right after the other in the queue. If the time it takes to transmit a packet on the router's outbound link is less than 20 msecs, then the spacing between first and second packets becomes less than 20 msecs.

The situation is analogous to driving cars on roads. Suppose you and your friend are each driving in your own cars from San Diego to Phoenix. Suppose you and your

friend have similar driving styles, and that you both drive at 100 km/hour, traffic permitting. If your friend starts out one hour before you, depending on intervening traffic, you may arrive at Phoenix more or less than one hour after your friend.

If the receiver ignores the presence of jitter and plays out chunks as soon as they arrive, then the resulting audio quality can easily become unintelligible at the receiver. Fortunately, jitter can often be removed by using **sequence numbers**, **timestamps**, and a **playout delay**, as discussed below.

9.3.2 Removing Jitter at the Receiver for Audio

For our VoIP application, where packets are being generated periodically, the receiver should attempt to provide periodic playout of voice chunks in the presence of random network jitter. This is typically done by combining the following two mechanisms:

- *Prepending each chunk with a* ***timestamp***. The sender stamps each chunk with the time at which the chunk was generated.
- ***Delaying playout*** *of chunks at the receiver*. As we saw in our earlier discussion of Figure 9.1, the playout delay of the received audio chunks must be long enough so that most of the packets are received before their scheduled playout times. This playout delay can either be fixed throughout the duration of the audio session or vary adaptively during the audio session lifetime.

We now discuss how these three mechanisms, when combined, can alleviate or even eliminate the effects of jitter. We examine two playback strategies: fixed playout delay and adaptive playout delay.

Fixed Playout Delay

With the fixed-delay strategy, the receiver attempts to play out each chunk exactly q msecs after the chunk is generated. So if a chunk is timestamped at the sender at time t, the receiver plays out the chunk at time $t + q$, assuming the chunk has arrived by that time. Packets that arrive after their scheduled playout times are discarded and considered lost.

What is a good choice for q? VoIP can support delays up to about 400 msecs, although a more satisfying conversational experience is achieved with smaller values of q. On the other hand, if q is made much smaller than 400 msecs, then many packets may miss their scheduled playback times due to the network-induced packet jitter. Roughly speaking, if large variations in end-to-end delay are typical, it is preferable to use a large q; on the other hand, if delay is small and variations in delay are also small, it is preferable to use a small q, perhaps less than 150 msecs.

The trade-off between the playback delay and packet loss is illustrated in Figure 9.4. The figure shows the times at which packets are generated and played

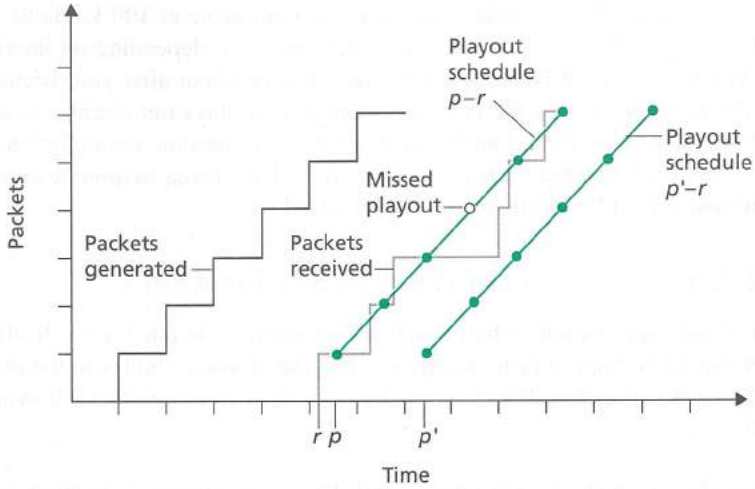


Figure 9.4 ♦ Packet loss for different fixed playout delays

out for a single talk spurt. Two distinct initial playout delays are considered. As shown by the leftmost staircase, the sender generates packets at regular intervals—say, every 20 msecs. The first packet in this talk spurt is received at time r. As shown in the figure, the arrivals of subsequent packets are not evenly spaced due to the network jitter.

For the first playout schedule, the fixed initial playout delay is set to $p - r$. With this schedule, the fourth packet does not arrive by its scheduled playout time, and the receiver considers it lost. For the second playout schedule, the fixed initial playout delay is set to $p' - r$. For this schedule, all packets arrive before their scheduled playout times, and there is therefore no loss.

Adaptive Playout Delay

The previous example demonstrates an important delay-loss trade-off that arises when designing a playout strategy with fixed playout delays. By making the initial playout delay large, most packets will make their deadlines and there will therefore be negligible loss; however, for conversational services such as VoIP, long delays can become bothersome if not intolerable. Ideally, we would like the playout delay to be minimized subject to the constraint that the loss be below a few percent.

The natural way to deal with this trade-off is to estimate the network delay and the variance of the network delay, and to adjust the playout delay accordingly at the beginning of each talk spurt. This adaptive adjustment of playout delays at the beginning of the talk spurts will cause the sender's silent periods to be compressed and elongated; however, compression and elongation of silence by a small amount is not noticeable in speech.

Following [Ramjee 1994], we now describe a generic algorithm that the receiver can use to adaptively adjust its playout delays. To this end, let

t_i = the timestamp of the ith packet = the time the packet was generated by the sender

r_i = the time packet i is received by receiver

p_i = the time packet i is played at receiver

The end-to-end network delay of the ith packet is $r_i - t_i$. Due to network jitter, this delay will vary from packet to packet. Let d_i denote an estimate of the *average* network delay upon reception of the ith packet. This estimate is constructed from the timestamps as follows:

$$d_i = (1 - u)\, d_{i-1} + u\,(r_i - t_i)$$

where u is a fixed constant (for example, $u = 0.01$). Thus d_i is a smoothed average of the observed network delays $r_1 - t_1, \ldots, r_i - t_i$. The estimate places more weight on the recently observed network delays than on the observed network delays of the distant past. This form of estimate should not be completely unfamiliar; a similar idea is used to estimate round-trip times in TCP, as discussed in Chapter 3. Let v_i denote an estimate of the average deviation of the delay from the estimated average delay. This estimate is also constructed from the timestamps:

$$v_i = (1 - u)\, v_{i-1} + u\,|r_i - t_i - d_i|$$

The estimates d_i and v_i are calculated for every packet received, although they are used only to determine the playout point for the first packet in any talk spurt.

Once having calculated these estimates, the receiver employs the following algorithm for the playout of packets. If packet i is the first packet of a talk spurt, its playout time, p_i, is computed as:

$$p_i = t_i + d_i + Kv_i$$

where K is a positive constant (for example, $K = 4$). The purpose of the Kv_i term is to set the playout time far enough into the future so that only a small fraction of the arriving packets in the talk spurt will be lost due to late arrivals. The playout point for any subsequent packet in a talk spurt is computed as an offset from the point in time when the first packet in the talk spurt was played out. In particular, let

$$q_i = p_i - t_i$$

be the length of time from when the first packet in the talk spurt is generated until it is played out. If packet j also belongs to this talk spurt, it is played out at time

$$p_j = t_j + q_i$$

The algorithm just described makes perfect sense assuming that the receiver can tell whether a packet is the first packet in the talk spurt. This can be done by examining the signal energy in each received packet.

9.3.3 Recovering from Packet Loss

We have discussed in some detail how a VoIP application can deal with packet jitter. We now briefly describe several schemes that attempt to preserve acceptable audio quality in the presence of packet loss. Such schemes are called **loss recovery schemes**. Here we define packet loss in a broad sense: A packet is lost either if it never arrives at the receiver or if it arrives after its scheduled playout time. Our VoIP example will again serve as a context for describing loss recovery schemes.

As mentioned at the beginning of this section, retransmitting lost packets may not be feasible in a real-time conversational application such as VoIP. Indeed, retransmitting a packet that has missed its playout deadline serves absolutely no purpose. And retransmitting a packet that overflowed a router queue cannot normally be accomplished quickly enough. Because of these considerations, VoIP applications often use some type of loss anticipation scheme. Two types of loss anticipation schemes are **forward error correction (FEC) and interleaving**.

Forward Error Correction (FEC)

The basic idea of FEC is to add redundant information to the original packet stream. For the cost of marginally increasing the transmission rate, the redundant information can be used to reconstruct approximations or exact versions of some of the lost packets. Following [Bolot 1996] and [Perkins 1998], we now outline two simple FEC mechanisms. The first mechanism sends a redundant encoded chunk after every n chunks. The redundant chunk is obtained by exclusive OR-ing the n original chunks [Shacham 1990]. In this manner if any one packet of the group of $n + 1$ packets is lost, the receiver can fully reconstruct the lost packet. But if two or more packets in a group are lost, the receiver cannot reconstruct the lost packets. By keeping $n + 1$, the group size, small, a large fraction of the lost packets can be recovered when loss is not excessive. However, the smaller the group size, the greater the relative increase of the transmission rate. In particular, the transmission rate increases by a factor of $1/n$, so that, if $n = 3$, then the transmission rate increases by 33 percent. Furthermore, this simple scheme increases the playout delay, as the receiver must wait to receive the entire group of packets before it can

begin playout. For more practical details about how FEC works for multimedia transport see [RFC 5109].

The second FEC mechanism is to send a lower-resolution audio stream as the redundant information. For example, the sender might create a nominal audio stream and a corresponding low-resolution, low-bit rate audio stream. (The nominal stream could be a PCM encoding at 64 kbps, and the lower-quality stream could be a GSM encoding at 13 kbps.) The low-bit rate stream is referred to as the redundant stream. As shown in Figure 9.5, the sender constructs the nth packet by taking the nth chunk from the nominal stream and appending to it the $(n - 1)$st chunk from the redundant stream. In this manner, whenever there is nonconsecutive packet loss, the receiver can conceal the loss by playing out the low-bit rate encoded chunk that arrives with the subsequent packet. Of course, low-bit rate chunks give lower quality than the nominal chunks. However, a stream of mostly high-quality chunks, occasional low-quality chunks, and no missing chunks gives good overall audio quality. Note that in this scheme, the receiver only has to receive two packets before playback, so that the increased playout delay is small. Furthermore, if the low-bit rate encoding is much less than the nominal encoding, then the marginal increase in the transmission rate will be small.

In order to cope with consecutive loss, we can use a simple variation. Instead of appending just the $(n - 1)$st low-bit rate chunk to the nth nominal chunk, the sender can append the $(n - 1)$st and $(n - 2)$nd low-bit rate chunk, or append the $(n - 1)$st and $(n - 3)$rd low-bit rate chunk, and so on. By appending more low-bit rate chunks to each nominal chunk, the audio quality at the receiver becomes acceptable for a wider variety of harsh best-effort environments. On the other hand, the additional chunks increase the transmission bandwidth and the playout delay.

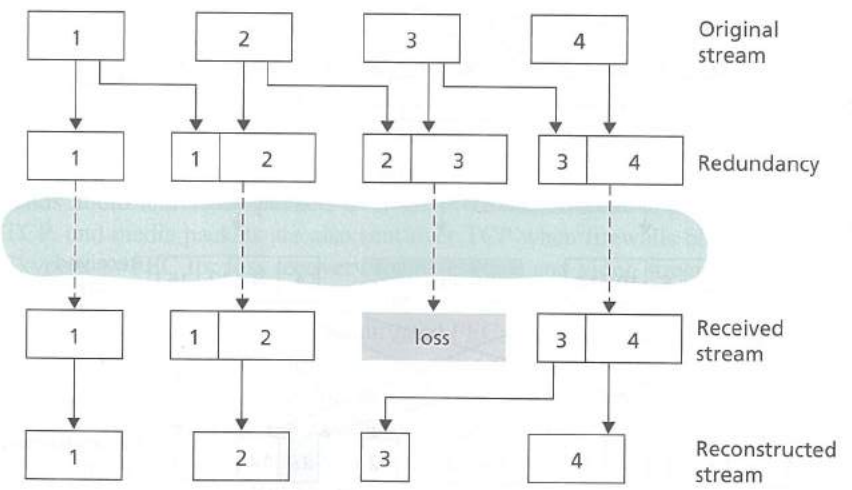


Figure 9.5 ♦ Piggybacking lower-quality redundant information

Interleaving

As an alternative to redundant transmission, a VoIP application can send interleaved audio. As shown in Figure 9.6, the sender resequences units of audio data before transmission, so that originally adjacent units are separated by a certain distance in the transmitted stream. Interleaving can mitigate the effect of packet losses. If, for example, units are 5 msecs in length and chunks are 20 msecs (that is, four units per chunk), then the first chunk could contain units 1, 5, 9, and 13; the second chunk could contain units 2, 6, 10, and 14; and so on. Figure 9.6 shows that the loss of a single packet from an interleaved stream results in multiple small gaps in the reconstructed stream, as opposed to the single large gap that would occur in a noninterleaved stream.

Interleaving can significantly improve the perceived quality of an audio stream [Perkins 1998]. It also has low overhead. The obvious disadvantage of interleaving is that it increases latency. This limits its use for conversational applications such as VoIP, although it can perform well for streaming stored audio. A major advantage of interleaving is that it does not increase the bandwidth requirements of a stream.

Error Concealment

Error concealment schemes attempt to produce a replacement for a lost packet that is similar to the original. As discussed in [Perkins 1998], this is possible since audio

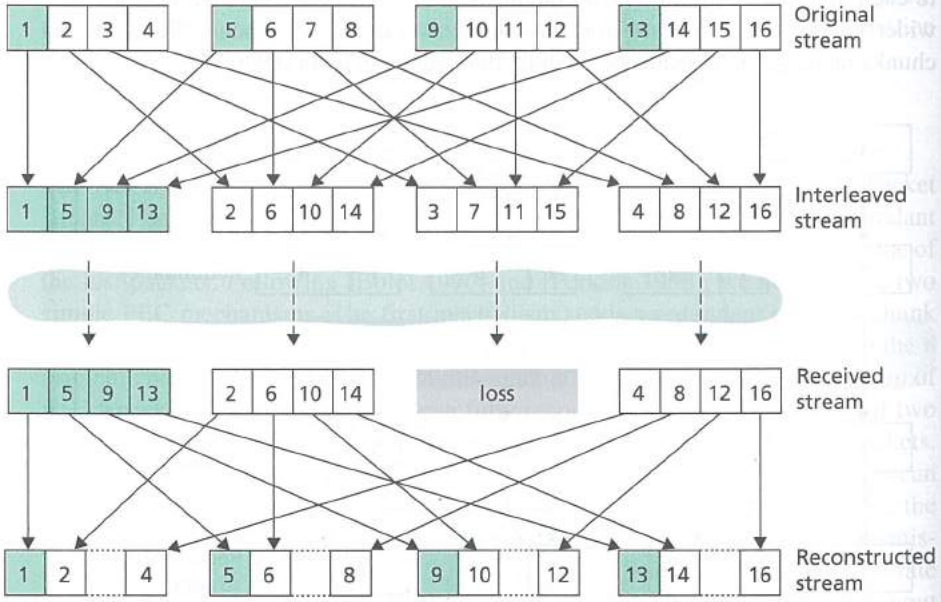


Figure 9.6 ♦ Sending interleaved audio

signals, and in particular speech, exhibit large amounts of short-term self-similarity. As such, these techniques work for relatively small loss rates (less than 15 percent), and for small packets (4–40 msecs). When the loss length approaches the length of a phoneme (5–100 msecs) these techniques break down, since whole phonemes may be missed by the listener.

Perhaps the simplest form of receiver-based recovery is packet repetition. Packet repetition replaces lost packets with copies of the packets that arrived immediately before the loss. It has low computational complexity and performs reasonably well. Another form of receiver-based recovery is interpolation, which uses audio before and after the loss to interpolate a suitable packet to cover the loss. Interpolation performs somewhat better than packet repetition but is significantly more computationally intensive [Perkins 1998].

9.3.4 Case Study: VoIP with Skype

Skype is an immensely popular VoIP application with over 50 million accounts active on a daily basis. In addition to providing host-to-host VoIP service, Skype offers host-to-phone services, phone-to-host services, and multi-party host-to-host video conferencing services. (Here, a host is again any Internet connected IP device, including PCs, tablets, and smartphones.) Skype was acquired by Microsoft in 2011.

Because the Skype protocol is proprietary, and because all Skype's control and media packets are encrypted, it is difficult to precisely determine how Skype operates. Nevertheless, from the Skype Web site and several measurement studies, researchers have learned how Skype generally works [Baset 2006; Guha 2006; Chen 2006; Suh 2006; Ren 2006; Zhang X 2012]. For both voice and video, the Skype clients have at their disposal many different codecs, which are capable of encoding the media at a wide range of rates and qualities. For example, video rates for Skype have been measured to be as low as 30 kbps for a low-quality session up to almost 1 Mbps for a high quality session [Zhang X 2012]. Typically, Skype's audio quality is better than the "POTS" (Plain Old Telephone Service) quality provided by the wire-line phone system. (Skype codecs typically sample voice at 16,000 samples/sec or higher, which provides richer tones than POTS, which samples at 8,000/sec.) By default, Skype sends audio and video packets over UDP. However, control packets are sent over TCP, and media packets are also sent over TCP when firewalls block UDP streams. Skype uses FEC for loss recovery for both voice and video streams sent over UDP. The Skype client also adapts the audio and video streams it sends to current network conditions, by changing video quality and FEC overhead [Zhang X 2012].

Skype uses P2P techniques in a number of innovative ways, nicely illustrating how P2P can be used in applications that go beyond content distribution and file sharing. As with instant messaging, host-to-host Internet telephony is inherently P2P since, at the heart of the application, pairs of users (that is, peers) communicate with each other in real time. But Skype also employs P2P techniques for two other important functions, namely, for user location and for NAT traversal.

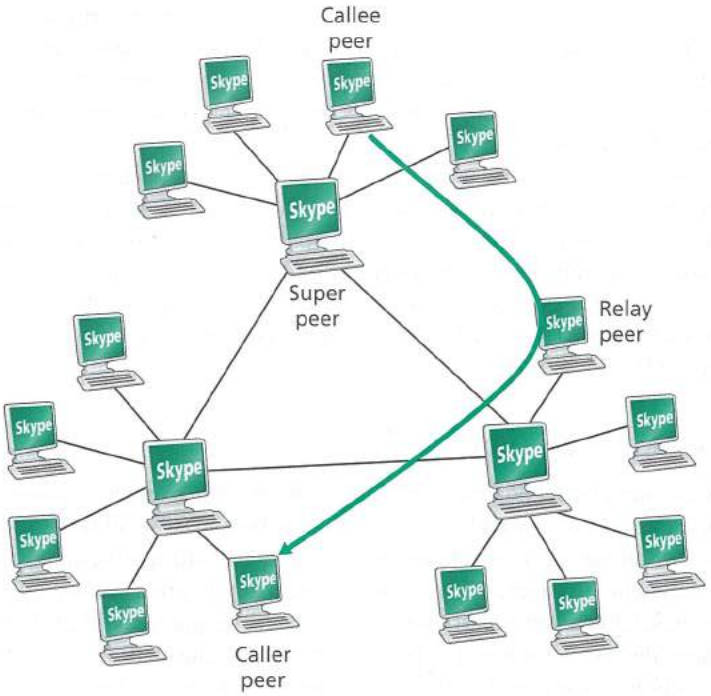


Figure 9.7 ♦ Skype peers

As shown in Figure 9.7, the peers (hosts) in Skype are organized into a hierarchical overlay network, with each peer classified as a super peer or an ordinary peer. Skype maintains an index that maps Skype usernames to current IP addresses (and port numbers). This index is distributed over the super peers. When Alice wants to call Bob, her Skype client searches the distributed index to determine Bob's current IP address. Because the Skype protocol is proprietary, it is currently not known how the index mappings are organized across the super peers, although some form of DHT organization is very possible.

P2P techniques are also used in Skype **relays**, which are useful for establishing calls between hosts in home networks. Many home network configurations provide access to the Internet through NATs, as discussed in Chapter 4. Recall that a NAT prevents a host from outside the home network from initiating a connection to a host within the home network. If *both* Skype callers have NATs, then there is a problem—neither can accept a call initiated by the other, making a call seemingly impossible. The clever use of super peers and relays nicely solves this problem. Suppose that when Alice signs in, she is assigned to a non-NATed super peer and initiates a session to that super peer. (Since Alice is initiating the session, her NAT permits this session.) This session allows Alice and her super peer to exchange

control messages. The same happens for Bob when he signs in. Now, when Alice wants to call Bob, she informs her super peer, who in turn informs Bob's super peer, who in turn informs Bob of Alice's incoming call. If Bob accepts the call, the two super peers select a third non-NATed super peer—the relay peer—whose job will be to relay data between Alice and Bob. Alice's and Bob's super peers then instruct Alice and Bob respectively to initiate a session with the relay. As shown in Figure 9.7, Alice then sends voice packets to the relay over the Alice-to-relay connection (which was initiated by Alice), and the relay then forwards these packets over the relay-to-Bob connection (which was initiated by Bob); packets from Bob to Alice flow over these same two relay connections in reverse. And *voila!*—Bob and Alice have an end-to-end connection even though neither can accept a session originating from outside.

Up to now, our discussion on Skype has focused on calls involving two persons. Now let's examine multi-party audio conference calls. With $N > 2$ participants, if each user were to send a copy of its audio stream to each of the $N - 1$ other users, then a total of $N(N - 1)$ audio streams would need to be sent into the network to support the audio conference. To reduce this bandwidth usage, Skype employs a clever distribution technique. Specifically, each user sends its audio stream to the conference initiator. The conference initiator combines the audio streams into one stream (basically by adding all the audio signals together) and then sends a copy of each combined stream to each of the other $N - 1$ participants. In this manner, the number of streams is reduced to $2(N - 1)$. For ordinary two-person video conversations, Skype routes the call peer-to-peer, unless NAT traversal is required, in which case the call is relayed through a non-NATed peer, as described earlier. For a video conference call involving $N > 2$ participants, due to the nature of the video medium, Skype does not combine the call into one stream at one location and then redistribute the stream to all the participants, as it does for voice calls. Instead, each participant's video stream is routed to a server cluster (located in Estonia as of 2011), which in turn relays to each participant the $N - 1$ streams of the $N - 1$ other participants [Zhang X 2012]. You may be wondering why each participant sends a copy to a server rather than directly sending a copy of its video stream to each of the other $N - 1$ participants? Indeed, for both approaches, $N(N - 1)$ video streams are being collectively received by the N participants in the conference. The reason is, because upstream link bandwidths are significantly lower than downstream link bandwidths in most access links, the upstream links may not be able to support the $N - 1$ streams with the P2P approach.

VoIP systems such as Skype, WeChat, and Google Talk introduce new privacy concerns. Specifically, when Alice and Bob communicate over VoIP, Alice can sniff Bob's IP address and then use geo-location services [MaxMind 2016; Quova 2016] to determine Bob's current location and ISP (for example, his work or home ISP). In fact, with Skype it is possible for Alice to block the transmission of certain packets during call establishment so that she obtains Bob's current IP address, say every hour, without Bob knowing that he is being tracked and without being on Bob's

contact list. Furthermore, the IP address discovered from Skype can be correlated with IP addresses found in BitTorrent, so that Alice can determine the files that Bob is downloading [LeBlond 2011]. Moreover, it is possible to partially decrypt a Skype call by doing a traffic analysis of the packet sizes in a stream [White 2011].

9.4 Protocols for Real-Time Conversational Applications

Real-time conversational applications, including VoIP and video conferencing, are compelling and very popular. It is therefore not surprising that standards bodies, such as the IETF and ITU, have been busy for many years (and continue to be busy!) at hammering out standards for this class of applications. With the appropriate standards in place for real-time conversational applications, independent companies are creating new products that interoperate with each other. In this section we examine RTP and SIP for real-time conversational applications. Both standards are enjoying widespread implementation in industry products.

9.4.1 RTP

In the previous section, we learned that the sender side of a VoIP application appends header fields to the audio chunks before passing them to the transport layer. These header fields include sequence numbers and timestamps. Since most multimedia networking applications can make use of sequence numbers and timestamps, it is convenient to have a standardized packet structure that includes fields for audio/video data, sequence number, and timestamp, as well as other potentially useful fields. RTP, defined in RFC 3550, is such a standard. RTP can be used for transporting common formats such as PCM, ACC, and MP3 for sound and MPEG and H.263 for video. It can also be used for transporting proprietary sound and video formats. Today, RTP enjoys widespread implementation in many products and research prototypes. It is also complementary to other important real-time interactive protocols, such as SIP.

In this section, we provide an introduction to RTP. We also encourage you to visit Henning Schulzrinne's RTP site [Schulzrinne-RTP 2012], which provides a wealth of information on the subject. Also, you may want to visit the RAT site [RAT 2012], which documents VoIP application that uses RTP.

RTP Basics

RTP typically runs on top of UDP. The sending side encapsulates a media chunk within an RTP packet, then encapsulates the packet in a UDP segment, and then

hands the segment to IP. The receiving side extracts the RTP packet from the UDP segment, then extracts the media chunk from the RTP packet, and then passes the chunk to the media player for decoding and rendering.

As an example, consider the use of RTP to transport voice. Suppose the voice source is PCM-encoded (that is, sampled, quantized, and digitized) at 64 kbps. Further suppose that the application collects the encoded data in 20-msec chunks, that is, 160 bytes in a chunk. The sending side precedes each chunk of the audio data with an **RTP header** that includes the type of audio encoding, a sequence number, and a timestamp. The RTP header is normally 12 bytes. The audio chunk along with the RTP header form the **RTP packet**. The RTP packet is then sent into the UDP socket interface. At the receiver side, the application receives the RTP packet from its socket interface. The application extracts the audio chunk from the RTP packet and uses the header fields of the RTP packet to properly decode and play back the audio chunk.

If an application incorporates RTP—instead of a proprietary scheme to provide payload type, sequence numbers, or timestamps—then the application will more easily interoperate with other networked multimedia applications. For example, if two different companies develop VoIP software and they both incorporate RTP into their product, there may be some hope that a user using one of the VoIP products will be able to communicate with a user using the other VoIP product. In Section 9.4.2, we'll see that RTP is often used in conjunction with SIP, an important standard for Internet telephony.

It should be emphasized that RTP does not provide any mechanism to ensure timely delivery of data or provide other quality-of-service (QoS) guarantees; it does not even guarantee delivery of packets or prevent out-of-order delivery of packets. Indeed, RTP encapsulation is seen only at the end systems. Routers do not distinguish between IP datagrams that carry RTP packets and IP datagrams that don't.

RTP allows each source (for example, a camera or a microphone) to be assigned its own independent RTP stream of packets. For example, for a video conference between two participants, four RTP streams could be opened—two streams for transmitting the audio (one in each direction) and two streams for transmitting the video (again, one in each direction). However, many popular encoding techniques—including MPEG 1 and MPEG 2—bundle the audio and video into a single stream during the encoding process. When the audio and video are bundled by the encoder, then only one RTP stream is generated in each direction.

RTP packets are not limited to unicast applications. They can also be sent over one-to-many and many-to-many multicast trees. For a many-to-many multicast session, all of the session's senders and sources typically use the same multicast group for sending their RTP streams. RTP multicast streams belonging together, such as audio and video streams emanating from multiple senders in a video conference application, belong to an **RTP session**.

Payload type	Sequence number	Timestamp	Synchronization source identifier	Miscellaneous fields

Figure 9.8 ♦ RTP header fields

RTP Packet Header Fields

As shown in Figure 9.8, the four main RTP packet header fields are the payload type, sequence number, timestamp, and source identifier fields.

The payload type field in the RTP packet is 7 bits long. For an audio stream, the payload type field is used to indicate the type of audio encoding (for example, PCM, adaptive delta modulation, linear predictive encoding) that is being used. If a sender decides to change the encoding in the middle of a session, the sender can inform the receiver of the change through this payload type field. The sender may want to change the encoding in order to increase the audio quality or to decrease the RTP stream bit rate. Table 9.2 lists some of the audio payload types currently supported by RTP.

For a video stream, the payload type is used to indicate the type of video encoding (for example, motion JPEG, MPEG 1, MPEG 2, H.261). Again, the sender can change video encoding on the fly during a session. Table 9.3 lists some of the video payload types currently supported by RTP. The other important fields are the following:

- *Sequence number field.* The sequence number field is 16 bits long. The sequence number increments by one for each RTP packet sent, and may be used by the receiver to detect packet loss and to restore packet sequence. For example, if the receiver side of the application receives a stream of RTP packets with a gap between sequence numbers 86 and 89, then the receiver knows that packets 87 and 88 are missing. The receiver can then attempt to conceal the lost data.
- *Timestamp field.* The timestamp field is 32 bits long. It reflects the sampling instant of the first byte in the RTP data packet. As we saw in the preceding section, the receiver can use timestamps to remove packet jitter introduced in the network and to provide synchronous playout at the receiver. The timestamp is derived from a sampling clock at the sender. As an example, for audio the timestamp clock increments by one for each sampling period (for example, each 125 μsec for an 8 kHz sampling clock); if the audio application generates chunks consisting of 160 encoded samples, then the timestamp increases by 160 for each RTP packet when the source is active. The timestamp clock continues to increase at a constant rate even if the source is inactive.
- *Synchronization source identifier (SSRC).* The SSRC field is 32 bits long. It identifies the source of the RTP stream. Typically, each stream in an RTP session has a distinct SSRC. The SSRC is not the IP address of the sender, but instead is a number that the source assigns randomly when the new stream is started. The probability that two streams get assigned the same SSRC is very small. Should this happen, the two sources pick a new SSRC value.

Payload-Type Number	Audio Format	Sampling Rate	Rate
0	PCM μ-law	8 kHz	64 kbps
1	1016	8 kHz	4.8 kbps
3	GSM	8 kHz	13 kbps
7	LPC	8 kHz	2.4 kbps
9	G.722	16 kHz	48–64 kbps
14	MPEG Audio	90 kHz	—
15	G.728	8 kHz	16 kbps

Table 9.2 ♦ Audio payload types supported by RTP

Payload-Type Number	Video Format
26	Motion JPEG
31	H.261
32	MPEG 1 video
33	MPEG 2 video

Table 9.3 ♦ Some video payload types supported by RTP

9.4.2 SIP

The Session Initiation Protocol (SIP), defined in [RFC 3261; RFC 5411], is an open and lightweight protocol that does the following:

- It provides mechanisms for establishing calls between a caller and a callee over an IP network. It allows the caller to notify the callee that it wants to start a call. It allows the participants to agree on media encodings. It also allows participants to end calls.
- It provides mechanisms for the caller to determine the current IP address of the callee. Users do not have a single, fixed IP address because they may be assigned addresses dynamically (using DHCP) and because they may have multiple IP devices, each with a different IP address.
- It provides mechanisms for call management, such as adding new media streams during the call, changing the encoding during the call, inviting new participants during the call, call transfer, and call holding.

Setting Up a Call to a Known IP Address

To understand the essence of SIP, it is best to take a look at a concrete example. In this example, Alice is at her PC and she wants to call Bob, who is also working at his PC. Alice's and Bob's PCs are both equipped with SIP-based software for making and receiving phone calls. In this initial example, we'll assume that Alice knows the IP address of Bob's PC. Figure 9.9 illustrates the SIP call-establishment process.

In Figure 9.9, we see that an SIP session begins when Alice sends Bob an INVITE message, which resembles an HTTP request message. This INVITE message is sent over UDP to the well-known port 5060 for SIP. (SIP messages can also be sent over TCP.) The INVITE message includes an identifier for Bob (bob@193.64.210.89), an indication of Alice's current IP address, an indication that Alice desires to receive audio, which is to be encoded in format AVP 0 (PCM encoded μ-law) and

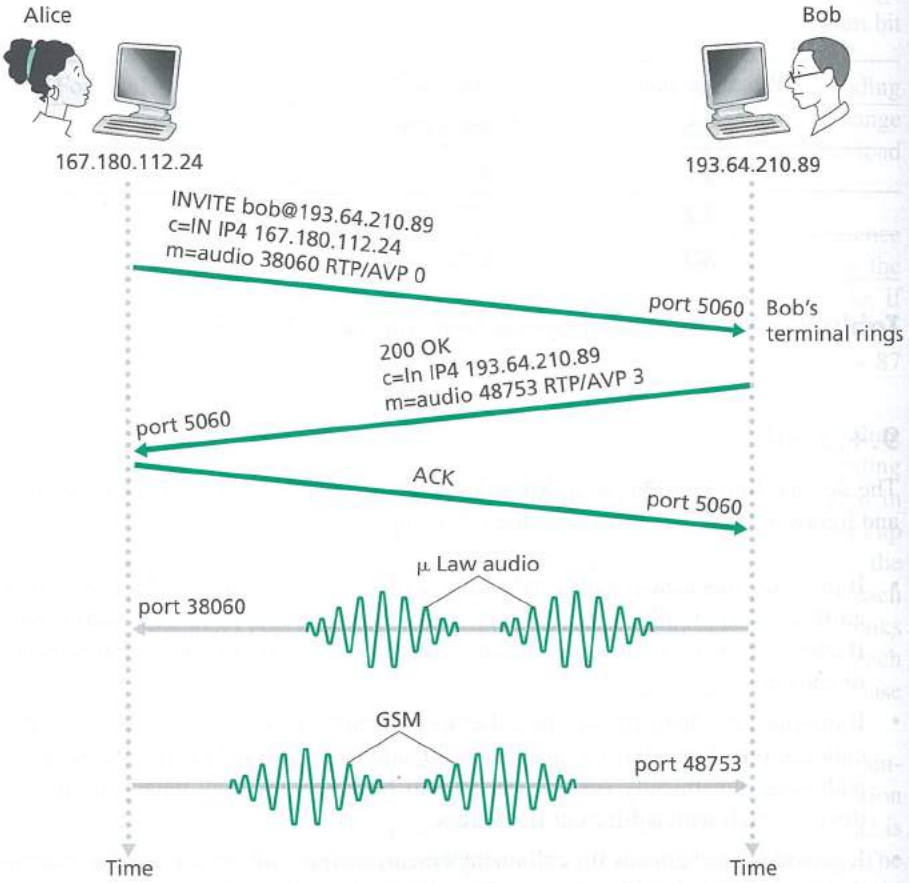


Figure 9.9 ♦ SIP call establishment when Alice knows Bob's IP address

encapsulated in RTP, and an indication that she wants to receive the RTP packets on port 38060. After receiving Alice's INVITE message, Bob sends an SIP response message, which resembles an HTTP response message. This response SIP message is also sent to the SIP port 5060. Bob's response includes a 200 OK as well as an indication of his IP address, his desired encoding and packetization for reception, and his port number to which the audio packets should be sent. Note that in this example Alice and Bob are going to use different audio-encoding mechanisms: Alice is asked to encode her audio with GSM whereas Bob is asked to encode his audio with PCM μ-law. After receiving Bob's response, Alice sends Bob an SIP acknowledgment message. After this SIP transaction, Bob and Alice can talk. (For visual convenience, Figure 9.9 shows Alice talking after Bob, but in truth they would normally talk at the same time.) Bob will encode and packetize the audio as requested and send the audio packets to port number 38060 at IP address 167.180.112.24. Alice will also encode and packetize the audio as requested and send the audio packets to port number 48753 at IP address 193.64.210.89.

From this simple example, we have learned a number of key characteristics of SIP. First, SIP is an out-of-band protocol: The SIP messages are sent and received in sockets that are different from those used for sending and receiving the media data. Second, the SIP messages themselves are ASCII-readable and resemble HTTP messages. Third, SIP requires all messages to be acknowledged, so it can run over UDP or TCP.

In this example, let's consider what would happen if Bob does not have a PCM μ-law codec for encoding audio. In this case, instead of responding with 200 OK, Bob would likely respond with a 606 Not Acceptable and list in the message all the codecs he can use. Alice would then choose one of the listed codecs and send another INVITE message, this time advertising the chosen codec. Bob could also simply reject the call by sending one of many possible rejection reply codes. (There are many such codes, including "busy," "gone," "payment required," and "forbidden.")

SIP Addresses

In the previous example, Bob's SIP address is sip:bob@193.64.210.89. However, we expect many—if not most—SIP addresses to resemble e-mail addresses. For example, Bob's address might be sip:bob@domain.com. When Alice's SIP device sends an INVITE message, the message would include this e-mail-like address; the SIP infrastructure would then route the message to the IP device that Bob is currently using (as we'll discuss below). Other possible forms for the SIP address could be Bob's legacy phone number or simply Bob's first/middle/last name (assuming it is unique).

An interesting feature of SIP addresses is that they can be included in Web pages, just as people's e-mail addresses are included in Web pages with the mailto URL. For example, suppose Bob has a personal homepage, and he wants to provide a means for visitors to the homepage to call him. He could then simply include the

URL sip:bob@domain.com. When the visitor clicks on the URL, the SIP application in the visitor's device is launched and an INVITE message is sent to Bob.

SIP Messages

In this short introduction to SIP, we'll not cover all SIP message types and headers. Instead, we'll take a brief look at the SIP INVITE message, along with a few common header lines. Let us again suppose that Alice wants to initiate a VoIP call to Bob, and this time Alice knows only Bob's SIP address, bob@domain.com, and does not know the IP address of the device that Bob is currently using. Then her message might look something like this:

```
INVITE sip:bob@domain.com SIP/2.0
Via: SIP/2.0/UDP 167.180.112.24
From: sip:alice@hereway.com
To: sip:bob@domain.com
Call-ID: a2e3a@pigeon.hereway.com
Content-Type: application/sdp
Content-Length: 885

c=IN IP4 167.180.112.24
m=audio 38060 RTP/AVP 0
```

The INVITE line includes the SIP version, as does an HTTP request message. Whenever an SIP message passes through an SIP device (including the device that originates the message), it attaches a Via header, which indicates the IP address of the device. (We'll see soon that the typical INVITE message passes through many SIP devices before reaching the callee's SIP application.) Similar to an e-mail message, the SIP message includes a From header line and a To header line. The message includes a Call-ID, which uniquely identifies the call (similar to the message-ID in e-mail). It includes a Content-Type header line, which defines the format used to describe the content contained in the SIP message. It also includes a Content-Length header line, which provides the length in bytes of the content in the message. Finally, after a carriage return and line feed, the message contains the content. In this case, the content provides information about Alice's IP address and how Alice wants to receive the audio.

Name Translation and User Location

In the example in Figure 9.9, we assumed that Alice's SIP device knew the IP address where Bob could be contacted. But this assumption is quite unrealistic, not only because IP addresses are often dynamically assigned with DHCP, but also because Bob may have multiple IP devices (for example, different devices for his home, work, and car). So now let us suppose that Alice knows only Bob's e-mail address,

bob@domain.com, and that this same address is used for SIP-based calls. In this case, Alice needs to obtain the IP address of the device that the user bob@domain.com is currently using. To find this out, Alice creates an INVITE message that begins with INVITE bob@domain.com SIP/2.0 and sends this message to an **SIP proxy**. The proxy will respond with an SIP reply that might include the IP address of the device that bob@domain.com is currently using. Alternatively, the reply might include the IP address of Bob's voicemail box, or it might include a URL of a Web page (that says "Bob is sleeping. Leave me alone!"). Also, the result returned by the proxy might depend on the caller: If the call is from Bob's wife, he might accept the call and supply his IP address; if the call is from Bob's mother-in-law, he might respond with the URL that points to the I-am-sleeping Web page!

Now, you are probably wondering, how can the proxy server determine the current IP address for bob@domain.com? To answer this question, we need to say a few words about another SIP device, the **SIP registrar**. Every SIP user has an associated registrar. Whenever a user launches an SIP application on a device, the application sends an SIP register message to the registrar, informing the registrar of its current IP address. For example, when Bob launches his SIP application on his PDA, the application would send a message along the lines of:

```
REGISTER sip:domain.com SIP/2.0
Via: SIP/2.0/UDP 193.64.210.89
From: sip:bob@domain.com
To: sip:bob@domain.com
Expires: 3600
```

Bob's registrar keeps track of Bob's current IP address. Whenever Bob switches to a new SIP device, the new device sends a new register message, indicating the new IP address. Also, if Bob remains at the same device for an extended period of time, the device will send refresh register messages, indicating that the most recently sent IP address is still valid. (In the example above, refresh messages need to be sent every 3600 seconds to maintain the address at the registrar server.) It is worth noting that the registrar is analogous to a DNS authoritative name server: The DNS server translates fixed host names to fixed IP addresses; the SIP registrar translates fixed human identifiers (for example, bob@domain.com) to dynamic IP addresses. Often SIP registrars and SIP proxies are run on the same host.

Now let's examine how Alice's SIP proxy server obtains Bob's current IP address. From the preceding discussion we see that the proxy server simply needs to forward Alice's INVITE message to Bob's registrar/proxy. The registrar/proxy could then forward the message to Bob's current SIP device. Finally, Bob, having now received Alice's INVITE message, could send an SIP response to Alice.

As an example, consider Figure 9.10, in which jim@umass.edu, currently working on 217.123.56.89, wants to initiate a Voice-over-IP (VoIP) session with keith@upenn.edu, currently working on 197.87.54.21. The following steps are taken:

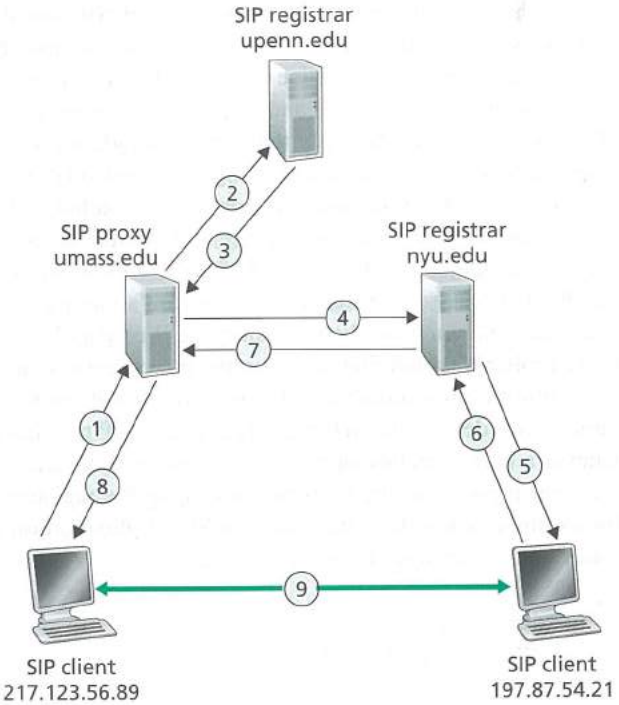


Figure 9.10 ♦ Session initiation, involving SIP proxies and registrars

(1) Jim sends an INVITE message to the umass SIP proxy. (2) The proxy does a DNS lookup on the SIP registrar upenn.edu (not shown in diagram) and then forwards the message to the registrar server. (3) Because keith@upenn.edu is no longer registered at the upenn registrar, the upenn registrar sends a redirect response, indicating that it should try keith@nyu.edu. (4) The umass proxy sends an INVITE message to the NYU SIP registrar. (5) The NYU registrar knows the IP address of keith@upenn.edu and forwards the INVITE message to the host 197.87.54.21, which is running Keith's SIP client. (6–8) An SIP response is sent back through registrars/proxies to the SIP client on 217.123.56.89. (9) Media is sent directly between the two clients. (There is also an SIP acknowledgment message, which is not shown.)

Our discussion of SIP has focused on call initiation for voice calls. SIP, being a signaling protocol for initiating and ending calls in general, can be used for video conference calls as well as for text-based sessions. In fact, SIP has become a fundamental component in many instant messaging applications. Readers desiring to learn more about SIP are encouraged to visit Henning Schulzrinne's SIP Web site [Schulzrinne-SIP 2016]. In particular, on this site you will find open source software for SIP clients and servers [SIP Software 2016].

9.5 Network Support for Multimedia

In Sections 9.2 through 9.4, we learned how application-level mechanisms such as client buffering, prefetching, adapting media quality to available bandwidth, adaptive playout, and loss mitigation techniques can be used by multimedia applications to improve a multimedia application's performance. We also learned how content distribution networks and P2P overlay networks can be used to provide a *system-level* approach for delivering multimedia content. These techniques and approaches are all designed to be used in today's best-effort Internet. Indeed, they are in use today precisely because the Internet provides only a single, best-effort class of service. But as designers of computer networks, we can't help but ask whether the *network* (rather than the applications or application-level infrastructure alone) might provide mechanisms to support multimedia content delivery. As we'll see shortly, the answer is, of course, "yes"! But we'll also see that a number of these new network-level mechanisms have yet to be widely deployed. This may be due to their complexity and to the fact that application-level techniques together with best-effort service and properly dimensioned network resources (for example, bandwidth) can indeed provide a "good-enough" (even if not-always-perfect) end-to-end multimedia delivery service.

Table 9.4 summarizes three broad approaches towards providing network-level support for multimedia applications.

- *Making the best of best-effort service.* The application-level mechanisms and infrastructure that we studied in Sections 9.2 through 9.4 can be successfully used in a well-dimensioned network where packet loss and excessive end-to-end delay rarely occur. When demand increases are forecasted, the ISPs deploy additional bandwidth and switching capacity to continue to ensure satisfactory delay and packet-loss performance [Huang 2005]. We'll discuss such **network dimensioning** further in Section 9.5.1.
- *Differentiated service.* Since the early days of the Internet, it's been envisioned that different types of traffic (for example, as indicated in the Type-of-Service field in the IP4v packet header) could be provided with different classes of service, rather than a single one-size-fits-all best-effort service. With **differentiated service**, one type of traffic might be given strict priority over another class of traffic when both types of traffic are queued at a router. For example, packets belonging to a real-time conversational application might be given priority over other packets due to their stringent delay constraints. Introducing differentiated service into the network will require new mechanisms for packet marking (indicating a packet's class of service), packet scheduling, and more. We'll cover differentiated service, and new network mechanisms needed to implement this service, in Sections 9.5.2 and 9.5.3.

Approach	Granularity	Guarantee	Mechanisms	Complexity	Deployment to date
Making the best of best-effort service	all traffic treated equally	none, or soft	application-layer support, CDNs, overlays, network-level resource provisioning	minimal	everywhere
Differentiated service	different classes of traffic treated differently	none, or soft	packet marking, policing, scheduling	medium	some
Per-connection Quality-of-Service (QoS) Guarantees	each source-destination flows treated differently	soft or hard, once flow is admitted	packet marking, policing, scheduling; call admission and signaling	light	little

Table 9.4 ♦ Three network-level approaches to supporting multimedia applications

- *Per-connection Quality-of-Service (QoS) Guarantees.* With per-connection QoS guarantees, each instance of an application explicitly reserves end-to-end bandwidth and thus has a guaranteed end-to-end performance. A **hard guarantee** means the application will receive its requested quality of service (QoS) with certainty. A **soft guarantee** means the application will receive its requested quality of service with high probability. For example, if a user wants to make a VoIP call from Host A to Host B, the user's VoIP application reserves bandwidth explicitly in each link along a route between the two hosts. But permitting applications to make reservations and requiring the network to honor the reservations requires some big changes. First, we need a protocol that, on behalf of the applications, reserves link bandwidth on the paths from the senders to their receivers. Second, we'll need new scheduling policies in the router queues so that per-connection bandwidth reservations can be honored. Finally, in order to make a reservation, the applications must give the network a description of the traffic that they intend to send into the network and the network will need to police each application's traffic to make sure that it abides by that description. These mechanisms, when combined, require new and complex software in hosts and routers. Because per-connection QoS guaranteed service has not seen significant deployment, we'll cover these mechanisms only briefly in Section 9.5.4.

9.5.1 Dimensioning Best-Effort Networks

Fundamentally, the difficulty in supporting multimedia applications arises from their stringent performance requirements—low end-to-end packet delay, delay jitter, and loss—and the fact that packet delay, delay jitter, and loss occur whenever the network becomes congested. A first approach to improving the quality of multimedia applications—an approach that can often be used to solve just about any problem where resources are constrained—is simply to "throw money at the problem" and thus simply avoid resource contention. In the case of networked multimedia, this means providing enough link capacity throughout the network so that network congestion, and its consequent packet delay and loss, never (or only very rarely) occurs. With enough link capacity, packets could zip through today's Internet without queuing delay or loss. From many perspectives this is an ideal situation—multimedia applications would perform perfectly, users would be happy, and this could all be achieved with no changes to Internet's best-effort architecture.

The question, of course, is how much capacity is "enough" to achieve this nirvana, and whether the costs of providing "enough" bandwidth are practical from a business standpoint to the ISPs. The question of how much capacity to provide at network links in a given topology to achieve a given level of performance is often known as **bandwidth provisioning**. The even more complicated problem of how to design a network topology (where to place routers, how to interconnect routers with links, and what capacity to assign to links) to achieve a given level of end-to-end performance is a network design problem often referred to as **network dimensioning**. Both bandwidth provisioning and network dimensioning are complex topics, well beyond the scope of this textbook. We note here, however, that the following issues must be addressed in order to predict application-level performance between two network end points, and thus provision enough capacity to meet an application's performance requirements.

- *Models of traffic demand between network end points.* Models may need to be specified at both the call level (for example, users "arriving" to the network and starting up end-to-end applications) and at the packet level (for example, packets being generated by ongoing applications). Note that workload may change over time.
- *Well-defined performance requirements.* For example, a performance requirement for supporting delay-sensitive traffic, such as a conversational multimedia application, might be that the probability that the end-to-end delay of the packet is greater than a maximum tolerable delay be less than some small value [Fraleigh 2003].
- *Models to predict end-to-end performance for a given workload model, and techniques to find a minimal cost bandwidth allocation that will result in all user*

requirements being met. Here, researchers are busy developing performance models that can quantify performance for a given workload, and optimization techniques to find minimal-cost bandwidth allocations meeting performance requirements.

Given that today's best-effort Internet could (from a technology standpoint) support multimedia traffic at an appropriate performance level if it were dimensioned to do so, the natural question is why today's Internet doesn't do so. The answers are primarily economic and organizational. From an economic standpoint, would users be willing to pay their ISPs enough for the ISPs to install sufficient bandwidth to support multimedia applications over a best-effort Internet? The organizational issues are perhaps even more daunting. Note that an end-to-end path between two multimedia end points will pass through the networks of multiple ISPs. From an organizational standpoint, would these ISPs be willing to cooperate (perhaps with revenue sharing) to ensure that the end-to-end path is properly dimensioned to support multimedia applications? For a perspective on these economic and organizational issues, see [Davies 2005]. For a perspective on provisioning tier-1 backbone networks to support delay-sensitive traffic, see [Fraleigh 2003].

9.5.2 Providing Multiple Classes of Service

Perhaps the simplest enhancement to the one-size-fits-all best-effort service in today's Internet is to divide traffic into classes, and provide different levels of service to these different classes of traffic. For example, an ISP might well want to provide a higher class of service to delay-sensitive Voice-over-IP or teleconferencing traffic (and charge more for this service!) than to elastic traffic such as e-mail or HTTP. Alternatively, an ISP may simply want to provide a higher quality of service to customers willing to pay more for this improved service. A number of residential wired-access ISPs and cellular wireless-access ISPs have adopted such tiered levels of service—with platinum-service subscribers receiving better performance than gold- or silver-service subscribers.

We're all familiar with different classes of service from our everyday lives—first-class airline passengers get better service than business-class passengers, who in turn get better service than those of us who fly economy class; VIPs are provided immediate entry to events while everyone else waits in line; elders are revered in some countries and provided seats of honor and the finest food at a table. It's important to note that such differential service is provided among aggregates of traffic, that is, among classes of traffic, not among individual connections. For example, all first-class passengers are handled the same (with no first-class passenger receiving any better treatment than any other first-class passenger), just as all VoIP packets would receive the same treatment within the network, independent of the particular end-to-end connection to which they belong. As we will see, by dealing with a small number of traffic aggregates, rather than a large number of individual connections,

the new network mechanisms required to provide better-than-best service can be kept relatively simple.

The early Internet designers clearly had this notion of multiple classes of service in mind. Recall the type-of-service (ToS) field in the IPv4 header discussed in Chapter 4. IEN123 [ISI 1979] describes the ToS field also present in an ancestor of the IPv4 datagram as follows: "The Type of Service [field] provides an indication of the abstract parameters of the quality of service desired. These parameters are to be used to guide the selection of the actual service parameters when transmitting a datagram through a particular network. Several networks offer service precedence, which somehow treats high precedence traffic as more important that other traffic." More than four decades ago, the vision of providing different levels of service to different classes of traffic was clear! However, it's taken us an equally long period of time to realize this vision.

Motivating Scenarios

Let's begin our discussion of network mechanisms for providing multiple classes of service with a few motivating scenarios.

Figure 9.11 shows a simple network scenario in which two application packet flows originate on Hosts H1 and H2 on one LAN and are destined for Hosts H3 and H4 on another LAN. The routers on the two LANs are connected by a 1.5 Mbps link. Let's assume the LAN speeds are significantly higher than 1.5 Mbps, and focus on the output queue of router R1; it is here that packet delay and packet loss will occur if the aggregate sending rate of H1 and H2 exceeds 1.5 Mbps. Let's further suppose that a 1 Mbps audio application (for example, a CD-quality audio call) shares the

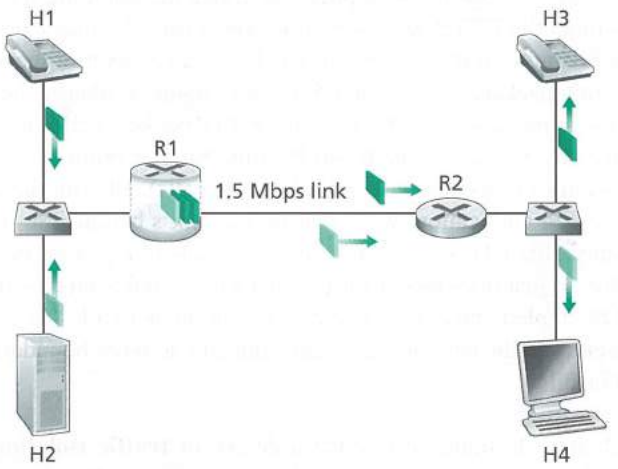


Figure 9.11 ♦ Competing audio and HTTP applications

1.5 Mbps link between R1 and R2 with an HTTP Web-browsing application that is downloading a Web page from H2 to H4.

In the best-effort Internet, the audio and HTTP packets are mixed in the output queue at R1 and (typically) transmitted in a first-in-first-out (FIFO) order. In this scenario, a burst of packets from the Web server could potentially fill up the queue, causing IP audio packets to be excessively delayed or lost due to buffer overflow at R1. How should we solve this potential problem? Given that the HTTP Web-browsing application does not have time constraints, our intuition might be to give strict priority to audio packets at R1. Under a strict priority scheduling discipline, an audio packet in the R1 output buffer would always be transmitted before any HTTP packet in the R1 output buffer. The link from R1 to R2 would look like a dedicated link of 1.5 Mbps to the audio traffic, with HTTP traffic using the R1-to-R2 link only when no audio traffic is queued. In order for R1 to distinguish between the audio and HTTP packets in its queue, each packet must be marked as belonging to one of these two classes of traffic. This was the original goal of the type-of-service (ToS) field in IPv4. As obvious as this might seem, this then is our first insight into mechanisms needed to provide multiple classes of traffic:

> **Insight 1: Packet marking** allows a router to distinguish among packets belonging to different classes of traffic.

Note that although our example considers a competing multimedia and elastic flow, the same insight applies to the case that platinum, gold, and silver classes of service are implemented—a packet-marking mechanism is still needed to indicate that class of service to which a packet belongs.

Now suppose that the router is configured to give priority to packets marked as belonging to the 1 Mbps audio application. Since the outgoing link speed is 1.5 Mbps, even though the HTTP packets receive lower priority, they can still, on average, receive 0.5 Mbps of transmission service. But what happens if the audio application starts sending packets at a rate of 1.5 Mbps or higher (either maliciously or due to an error in the application)? In this case, the HTTP packets will starve, that is, they will not receive any service on the R1-to-R2 link. Similar problems would occur if multiple applications (for example, multiple audio calls), all with the same class of service as the audio application, were sharing the link's bandwidth; they too could collectively starve the FTP session. Ideally, one wants a degree of isolation among classes of traffic so that one class of traffic can be protected from the other. This protection could be implemented at different places in the network—at each and every router, at first entry to the network, or at inter-domain network boundaries. This then is our second insight:

> **Insight 2:** It is desirable to provide a degree of **traffic isolation** among classes so that one class is not adversely affected by another class of traffic that misbehaves.

We'll examine several specific mechanisms for providing such isolation among traffic classes. We note here that two broad approaches can be taken. First, it is possible to perform **traffic policing**, as shown in Figure 9.12. If a traffic class or flow must meet certain criteria (for example, that the audio flow not exceed a peak rate of 1 Mbps), then a policing mechanism can be put into place to ensure that these criteria are indeed observed. If the policed application misbehaves, the policing mechanism will take some action (for example, drop or delay packets that are in violation of the criteria) so that the traffic actually entering the network conforms to the criteria. The leaky bucket mechanism that we'll examine shortly is perhaps the most widely used policing mechanism. In Figure 9.12, the packet classification and marking mechanism (Insight 1) and the policing mechanism (Insight 2) are both implemented together at the network's edge, either in the end system or at an edge router.

A complementary approach for providing isolation among traffic classes is for the link-level packet-scheduling mechanism to explicitly allocate a fixed amount of link bandwidth to each class. For example, the audio class could be allocated 1 Mbps at R1, and the HTTP class could be allocated 0.5 Mbps. In this case, the audio and

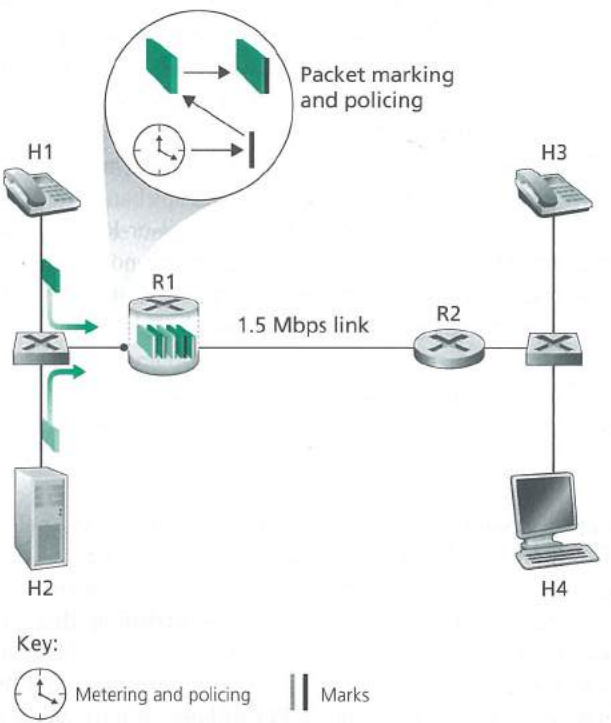


Figure 9.12 ♦ Policing (and marking) the audio and HTTP traffic classes

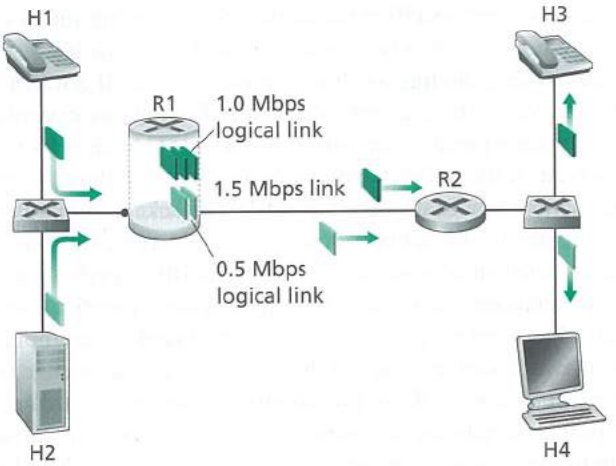


Figure 9.13 ♦ Logical isolation of audio and HTTP traffic classes

HTTP flows see a logical link with capacity 1.0 and 0.5 Mbps, respectively, as shown in Figure 9.13. With strict enforcement of the link-level allocation of bandwidth, a class can use only the amount of bandwidth that has been allocated; in particular, it cannot utilize bandwidth that is not currently being used by others. For example, if the audio flow goes silent (for example, if the speaker pauses and generates no audio packets), the HTTP flow would still not be able to transmit more than 0.5 Mbps over the R1-to-R2 link, even though the audio flow's 1 Mbps bandwidth allocation is not being used at that moment. Since bandwidth is a "use-it-or-lose-it" resource, there is no reason to prevent HTTP traffic from using bandwidth not used by the audio traffic. We'd like to use bandwidth as efficiently as possible, never wasting it when it could be otherwise used. This gives rise to our third insight:

> **Insight 3:** While providing isolation among classes or flows, it is desirable to use resources (for example, link bandwidth and buffers) as efficiently as possible.

Recall from our discussion in Sections 1.3 and 4.2 that packets belonging to various network flows are multiplexed and queued for transmission at the output buffers associated with a link. The manner in which queued packets are selected for transmission on the link is known as the **link-scheduling discipline**, and was discussed in detail in Section 4.2. Recall that in Section 4.2 three link-scheduling disciplines were discussed, namely, FIFO, priority queuing, and Weighted Fair Queuing (WFQ). We'll see soon see that WFQ will play a particularly important role for isolating the traffic classes.

The Leaky Bucket

One of our earlier insights was that policing, the regulation of the rate at which a class or flow (we will assume the unit of policing is a flow in our discussion below) is allowed to inject packets into the network, is an important QoS mechanism. But what aspects of a flow's packet rate should be policed? We can identify three important policing criteria, each differing from the other according to the time scale over which the packet flow is policed:

- *Average rate.* The network may wish to limit the long-term average rate (packets per time interval) at which a flow's packets can be sent into the network. A crucial issue here is the interval of time over which the average rate will be policed. A flow whose average rate is limited to 100 packets per second is more constrained than a source that is limited to 6,000 packets per minute, even though both have the same average rate over a long enough interval of time. For example, the latter constraint would allow a flow to send 1,000 packets in a given second-long interval of time, while the former constraint would disallow this sending behavior.
- *Peak rate.* While the average-rate constraint limits the amount of traffic that can be sent into the network over a relatively long period of time, a peak-rate constraint limits the maximum number of packets that can be sent over a shorter period of time. Using our example above, the network may police a flow at an average rate of 6,000 packets per minute, while limiting the flow's peak rate to 1,500 packets per second.
- *Burst size.* The network may also wish to limit the maximum number of packets (the "burst" of packets) that can be sent into the network over an extremely short interval of time. In the limit, as the interval length approaches zero, the burst size limits the number of packets that can be instantaneously sent into the network. Even though it is physically impossible to instantaneously send multiple packets into the network (after all, every link has a physical transmission rate that cannot be exceeded!), the abstraction of a maximum burst size is a useful one.

The leaky bucket mechanism is an abstraction that can be used to characterize these policing limits. As shown in Figure 9.14, a leaky bucket consists of a bucket that can hold up to b tokens. Tokens are added to this bucket as follows. New tokens, which may potentially be added to the bucket, are always being generated at a rate of r tokens per second. (We assume here for simplicity that the unit of time is a second.) If the bucket is filled with less than b tokens when a token is generated, the newly generated token is added to the bucket; otherwise the newly generated token is ignored, and the token bucket remains full with b tokens.

Let us now consider how the leaky bucket can be used to police a packet flow. Suppose that before a packet is transmitted into the network, it must first remove a token from the token bucket. If the token bucket is empty, the packet must wait for

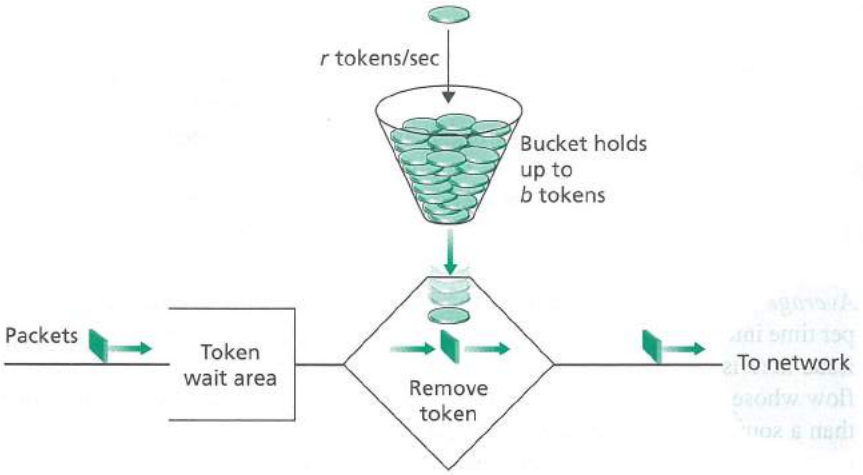


Figure 9.14 ♦ The leaky bucket policer

a token. (An alternative is for the packet to be dropped, although we will not consider that option here.) Let us now consider how this behavior polices a traffic flow. Because there can be at most b tokens in the bucket, the maximum burst size for a leaky-bucket-policed flow is b packets. Furthermore, because the token generation rate is r, the maximum number of packets that can enter the network of *any* interval of time of length t is $rt + b$. Thus, the token-generation rate, r, serves to limit the long-term average rate at which packets can enter the network. It is also possible to use leaky buckets (specifically, two leaky buckets in series) to police a flow's peak rate in addition to the long-term average rate; see the homework problems at the end of this chapter.

Leaky Bucket + Weighted Fair Queuing = Provable Maximum Delay in a Queue

Let's close our discussion on policing by showing how the leaky bucket and WFQ can be combined to provide a bound on the delay through a router's queue. (Readers who have forgotten about WFQ are encouraged to review WFQ, which is covered in Section 4.2.) Let's consider a router's output link that multiplexes n flows, each policed by a leaky bucket with parameters b_i and r_i, $i = 1, \ldots, n$, using WFQ scheduling. We use the term *flow* here loosely to refer to the set of packets that are not distinguished from each other by the scheduler. In practice, a flow might be comprised of traffic from a single end-to-end connection or a collection of many such connections, see Figure 9.15.

Recall from our discussion of WFQ that each flow, i, is guaranteed to receive a share of the link bandwidth equal to at least $R \cdot w_i/(\sum w_j)$, where R is the transmission

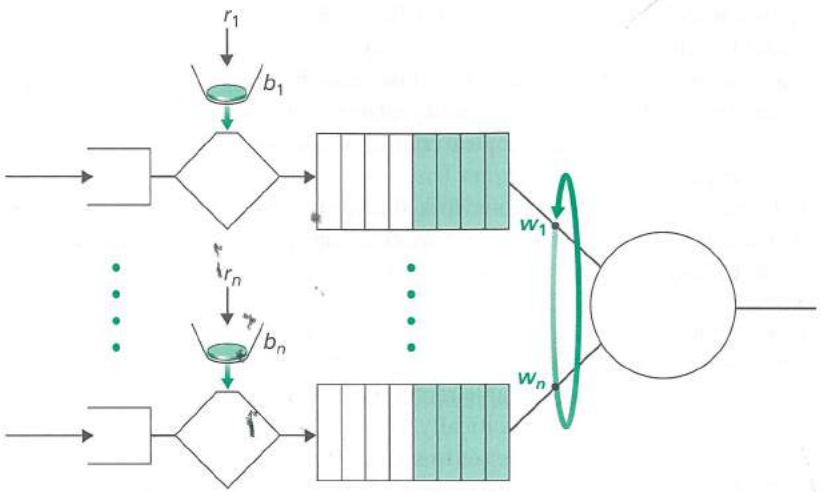


Figure 9.15 ♦ *n* multiplexed leaky bucket flows with WFQ scheduling

rate of the link in packets/sec. What then is the maximum delay that a packet will experience while waiting for service in the WFQ (that is, after passing through the leaky bucket)? Let us focus on flow 1. Suppose that flow 1's token bucket is initially full. A burst of b_1 packets then arrives to the leaky bucket policer for flow 1. These packets remove all of the tokens (without wait) from the leaky bucket and then join the WFQ waiting area for flow 1. Since these b_1 packets are served at a rate of at least $R \cdot w_i/(\sum w_j)$ packet/sec, the last of these packets will then have a maximum delay, $d_{\max}$, until its transmission is completed, where

$$d_{\max} = \frac{b_1}{R \cdot w_1/\sum w_j}$$

The rationale behind this formula is that if there are b_1 packets in the queue and packets are being serviced (removed) from the queue at a rate of at least $R \cdot w_1/(\sum w_j)$ packets per second, then the amount of time until the last bit of the last packet is transmitted cannot be more than $b_1/(R \cdot w_1/(\sum w_j))$. A homework problem asks you to prove that as long as $r_1 < R \cdot w_1/(\sum w_j)$, then $d_{\max}$ is indeed the maximum delay that any packet in flow 1 will ever experience in the WFQ queue.

9.5.3 Diffserv

Having seen the motivation, insights, and specific mechanisms for providing multiple classes of service, let's wrap up our study of approaches toward proving multiple classes of service with an example—the Internet Diffserv architecture [RFC 2475; Kilkki 1999]. Diffserv provides service differentiation—that is, the

ability to handle different classes of traffic in different ways within the Internet in a scalable manner. The need for scalability arises from the fact that millions of simultaneous source-destination traffic flows may be present at a backbone router. We'll see shortly that this need is met by placing only simple functionality within the network core, with more complex control operations being implemented at the network's edge.

Let's begin with the simple network shown in Figure 9.16. We'll describe one possible use of Diffserv here; other variations are possible, as described in RFC 2475. The Diffserv architecture consists of two sets of functional elements:

- *Edge functions: Packet classification and traffic conditioning.* At the incoming edge of the network (that is, at either a Diffserv-capable host that generates traffic or at the first Diffserv-capable router that the traffic passes through), arriving packets are marked. More specifically, the differentiated service (DS) field in the IPv4 or IPv6 packet header is set to some value [RFC 3260]. The definition of the DS field is intended to supersede the earlier definitions of the IPv4 type-of-service field and the IPv6 traffic class fields that we discussed in Chapter 4. For example, in Figure 9.16, packets being sent from H1 to H3 might be marked at R1, while packets being sent from H2 to H4 might be marked at R2. The mark that a packet receives identifies the class of traffic to which it belongs. Different classes of traffic will then receive different service within the core network.

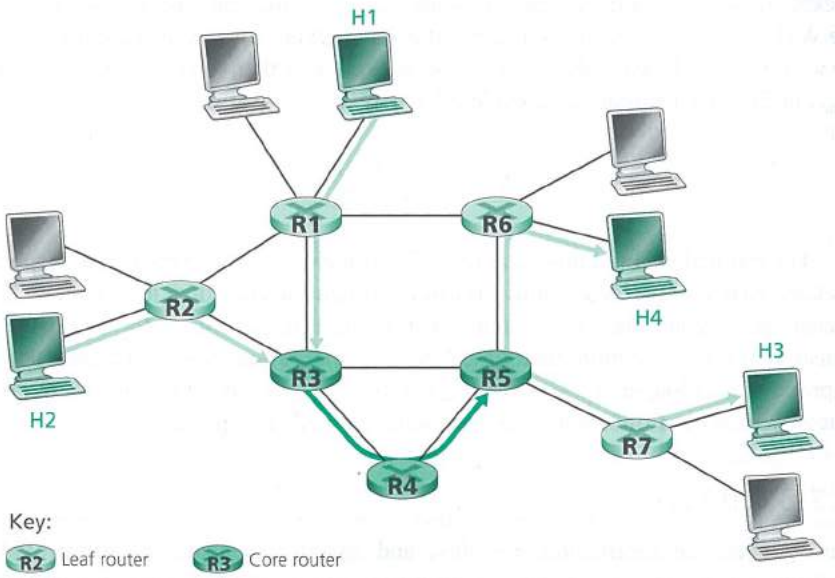


Figure 9.16 ♦ A simple Diffserv network example

- *Core function: Forwarding.* When a DS-marked packet arrives at a Diffserv-capable router, the packet is forwarded onto its next hop according to the so-called per-hop behavior (PHB) associated with that packet's class. The per-hop behavior influences how a router's buffers and link bandwidth are shared among the competing classes of traffic. A crucial tenet of the Diffserv architecture is that a router's per-hop behavior will be based only on packet markings, that is, the class of traffic to which a packet belongs. Thus, if packets being sent from H1 to H3 in Figure 9.16 receive the same marking as packets being sent from H2 to H4, then the network routers treat these packets as an aggregate, without distinguishing whether the packets originated at H1 or H2. For example, R3 would not distinguish between packets from H1 and H2 when forwarding these packets on to R4. Thus, the Diffserv architecture obviates the need to keep router state for individual source-destination pairs—a critical consideration in making Diffserv scalable.

An analogy might prove useful here. At many large-scale social events (for example, a large public reception, a large dance club or discothèque, a concert, or a football game), people entering the event receive a pass of one type or another: VIP passes for Very Important People; over-21 passes for people who are 21 years old or older (for example, if alcoholic drinks are to be served); backstage passes at concerts; press passes for reporters; even an ordinary pass for the Ordinary Person. These passes are typically distributed upon entry to the event, that is, at the edge of the event. It is here at the edge where computationally intensive operations, such as paying for entry, checking for the appropriate type of invitation, and matching an invitation against a piece of identification, are performed. Furthermore, there may be a limit on the number of people of a given type that are allowed into an event. If there is such a limit, people may have to wait before entering the event. Once inside the event, one's pass allows one to receive differentiated service at many locations around the event—a VIP is provided with free drinks, a better table, free food, entry to exclusive rooms, and fawning service. Conversely, an ordinary person is excluded from certain areas, pays for drinks, and receives only basic service. In both cases, the service received within the event depends solely on the type of one's pass. Moreover, all people within a class are treated alike.

Figure 9.17 provides a logical view of the classification and marking functions within the edge router. Packets arriving to the edge router are first classified. The classifier selects packets based on the values of one or more packet header fields (for example, source address, destination address, source port, destination port, and protocol ID) and steers the packet to the appropriate marking function. As noted above, a packet's marking is carried in the DS field in the packet header.

In some cases, an end user may have agreed to limit its packet-sending rate to conform to a declared **traffic profile**. The traffic profile might contain a limit on the peak rate, as well as the burstiness of the packet flow, as we saw previously with the leaky bucket mechanism. As long as the user sends packets into the network in a way that conforms to the negotiated traffic profile, the packets receive their priority

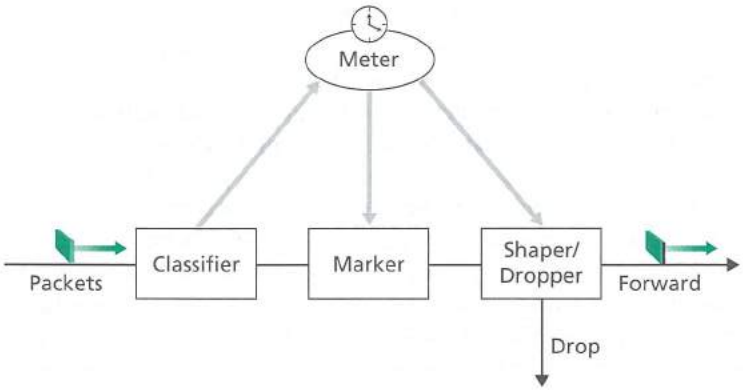


Figure 9.17 ♦ A simple Diffserv network example

marking and are forwarded along their route to the destination. On the other hand, if the traffic profile is violated, out-of-profile packets might be marked differently, might be shaped (for example, delayed so that a maximum rate constraint would be observed), or might be dropped at the network edge. The role of the **metering function**, shown in Figure 9.17, is to compare the incoming packet flow with the negotiated traffic profile and to determine whether a packet is within the negotiated traffic profile. The actual decision about whether to immediately remark, forward, delay, or drop a packet is a policy issue determined by the network administrator and is *not* specified in the Diffserv architecture.

So far, we have focused on the marking and policing functions in the Diffserv architecture. The second key component of the Diffserv architecture involves the per-hop behavior (PHB) performed by Diffserv-capable routers. PHB is rather cryptically, but carefully, defined as "a description of the externally observable forwarding behavior of a Diffserv node applied to a particular Diffserv behavior aggregate" [RFC 2475]. Digging a little deeper into this definition, we can see several important considerations embedded within:

- A PHB can result in different classes of traffic receiving different performance (that is, different externally observable forwarding behaviors).
- While a PHB defines differences in performance (behavior) among classes, it does not mandate any particular mechanism for achieving these behaviors. As long as the externally observable performance criteria are met, any implementation mechanism and any buffer/bandwidth allocation policy can be used. For example, a PHB would not require that a particular packet-queuing discipline (for example, a priority queue versus a WFQ queue versus a FCFS queue) be used to achieve a particular behavior. The PHB is the end, to which resource allocation and implementation mechanisms are the means.
- Differences in performance must be observable and hence measurable.

Two PHBs have been defined: an expedited forwarding (EF) PHB [RFC 3246] and an assured forwarding (AF) PHB [RFC 2597]. The **expedited forwarding** PHB specifies that the departure rate of a class of traffic from a router must equal or exceed a configured rate. The **assured forwarding** PHB divides traffic into four classes, where each AF class is guaranteed to be provided with some minimum amount of bandwidth and buffering.

Let's close our discussion of Diffserv with a few observations regarding its service model. First, we have implicitly assumed that Diffserv is deployed within a single administrative domain, but typically an end-to-end service must be fashioned from multiple ISPs sitting between communicating end systems. In order to provide end-to-end Diffserv service, all the ISPs between the end systems must not only provide this service, but most also cooperate and make settlements in order to offer end customers true end-to-end service. Without this kind of cooperation, ISPs directly selling Diffserv service to customers will find themselves repeatedly saying: "Yes, we know you paid extra, but we don't have a service agreement with the ISP that dropped and delayed your traffic. I'm sorry that there were so many gaps in your VoIP call!" Second, if Diffserv were actually in place and the network ran at only moderate load, most of the time there would be no perceived difference between a best-effort service and a Diffserv service. Indeed, end-to-end delay is usually dominated by access rates and router hops rather than by queuing delays in the routers. Imagine the unhappy Diffserv customer who has paid more for premium service but finds that the best-effort service being provided to others almost always has the same performance as premium service!

9.5.4 Per-Connection Quality-of-Service (QoS) Guarantees: Resource Reservation and Call Admission

In the previous section, we have seen that packet marking and policing, traffic isolation, and link-level scheduling can provide one class of service with better performance than another. Under certain scheduling disciplines, such as priority scheduling, the lower classes of traffic are essentially "invisible" to the highest-priority class of traffic. With proper network dimensioning, the highest class of service can indeed achieve extremely low packet loss and delay—essentially circuit-like performance. But can the network *guarantee* that an ongoing flow in a high-priority traffic class will continue to receive such service throughout the flow's duration using only the mechanisms that we have described so far? It cannot. In this section, we'll see why yet additional network mechanisms and protocols are required when a hard service guarantee is provided to individual connections.

Let's return to our scenario from Section 9.5.2 and consider two 1 Mbps audio applications transmitting their packets over the 1.5 Mbps link, as shown in Figure 9.18. The combined data rate of the two flows (2 Mbps) exceeds the link capacity. Even with classification and marking, isolation of flows, and sharing of unused bandwidth (of which there is none), this is clearly a losing proposition. There is simply not enough bandwidth to accommodate the needs of both applications at

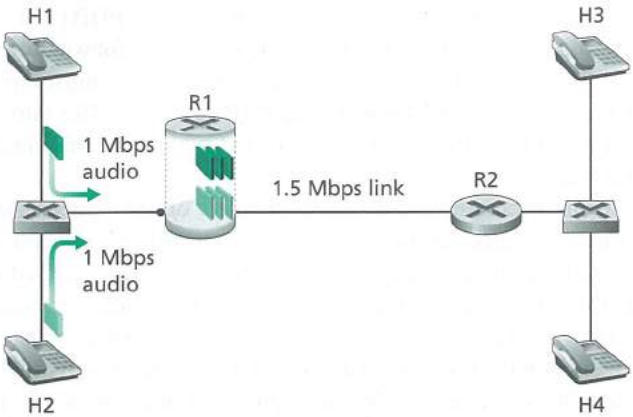


Figure 9.18 ♦ Two competing audio applications overloading the R1-to-R2 link

the same time. If the two applications equally share the bandwidth, each application would lose 25 percent of its transmitted packets. This is such an unacceptably low QoS that both audio applications are completely unusable; there's no need even to transmit any audio packets in the first place.

Given that the two applications in Figure 9.18 cannot both be satisfied simultaneously, what should the network do? Allowing both to proceed with an unusable QoS wastes network resources on application flows that ultimately provide no utility to the end user. The answer is hopefully clear—one of the application flows should be blocked (that is, denied access to the network), while the other should be allowed to proceed on, using the full 1 Mbps needed by the application. The telephone network is an example of a network that performs such call blocking—if the required resources (an end-to-end circuit in the case of the telephone network) cannot be allocated to the call, the call is blocked (prevented from entering the network) and a busy signal is returned to the user. In our example, there is no gain in allowing a flow into the network if it will not receive a sufficient QoS to be considered usable. Indeed, there is a cost to admitting a flow that does not receive its needed QoS, as network resources are being used to support a flow that provides no utility to the end user.

By explicitly admitting or blocking flows based on their resource requirements, and the source requirements of already-admitted flows, the network can guarantee that admitted flows will be able to receive their requested QoS. Implicit in the need to provide a guaranteed QoS to a flow is the need for the flow to declare its QoS requirements. This process of having a flow declare its QoS requirement, and then having the network either accept the flow (at the required QoS) or block the flow is referred to as the **call admission** process. This then is our fourth insight (in addition to the three earlier insights from Section 9.5.2,) into the mechanisms needed to provide QoS.

Insight 4: If sufficient resources will not always be available, and QoS is to be *guaranteed*, a call admission process is needed in which flows declare their QoS requirements and are then either admitted to the network (at the required QoS) or blocked from the network (if the required QoS cannot be provided by the network).

Our motivating example in Figure 9.18 highlights the need for several new network mechanisms and protocols if a call (an end-to-end flow) is to be guaranteed a given quality of service once it begins:

- *Resource reservation.* The only way to *guarantee* that a call will have the resources (link bandwidth, buffers) needed to meet its desired QoS is to explicitly allocate those resources to the call—a process known in networking parlance as **resource reservation**. Once resources are reserved, the call has on-demand access to these resources throughout its duration, regardless of the demands of all other calls. If a call reserves and receives a guarantee of x Mbps of link bandwidth, and never transmits at a rate greater than x, the call will see loss- and delay-free performance.
- *Call admission.* If resources are to be reserved, then the network must have a mechanism for calls to request and reserve resources. Since resources are not infinite, a call making a call admission request will be denied admission, that is, be blocked, if the requested resources are not available. Such a call admission is performed by the telephone network—we request resources when we dial a number. If the circuits (TDMA slots) needed to complete the call are available, the circuits are allocated and the call is completed. If the circuits are not available, then the call is blocked, and we receive a busy signal. A blocked call can try again to gain admission to the network, but it is not allowed to send traffic into the network until it has successfully completed the call admission process. Of course, a router that allocates link bandwidth should not allocate more than is available at that link. Typically, a call may reserve only a fraction of the link's bandwidth, and so a router may allocate link bandwidth to more than one call. However, the sum of the allocated bandwidth to all calls should be less than the link capacity if hard quality of service guarantees are to be provided.
- *Call setup signaling.* The call admission process described above requires that a call be able to reserve sufficient resources at each and every network router on its source-to-destination path to ensure that its end-to-end QoS requirement is met. Each router must determine the local resources required by the session, consider the amounts of its resources that are already committed to other ongoing sessions, and determine whether it has sufficient resources to satisfy the per-hop QoS requirement of the session at this router without violating local QoS guarantees made to an already-admitted session. A signaling protocol is needed to coordinate these various activities—the per-hop allocation of local resources, as well as the overall end-to-end decision of whether or not the call has been able to reserve suf-

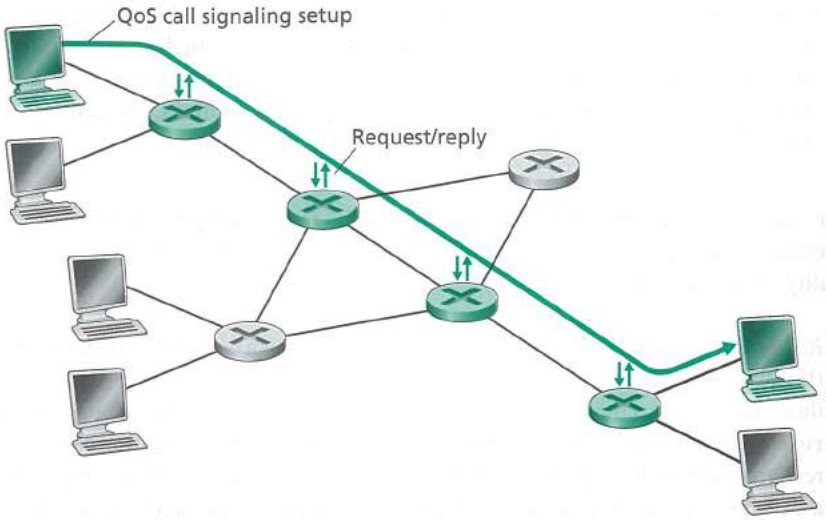


Figure 9.19 ♦ The call setup process

ficient resources at each and every router on the end-to-end path. This is the job of the **call setup protocol**, as shown in Figure 9.19. The **RSVP protocol** [Zhang 1993, RFC 2210] was proposed for this purpose within an Internet architecture for providing quality-of-service guarantees. In ATM networks, the Q2931b protocol [Black 1995] carries this information among the ATM network's switches and end point.

Despite a tremendous amount of research and development, and even products that provide for per-connection quality of service guarantees, there has been almost no extended deployment of such services. There are many possible reasons. First and foremost, it may well be the case that the simple application-level mechanisms that we studied in Sections 9.2 through 9.4, combined with proper network dimensioning (Section 9.5.1) provide "good enough" best-effort network service for multimedia applications. In addition, the added complexity and cost of deploying and managing a network that provides per-connection quality of service guarantees may be judged by ISPs to be simply too high given predicted customer revenues for that service.

9.6 Summary

Multimedia networking is one of the most exciting developments in the Internet today. People throughout the world less and less time in front of their televisions, and are instead use their smartphones and devices to receive audio and video trans-

missions, both live and prerecorded. Moreover, with sites like YouTube, users have become producers as well as consumers of multimedia Internet content. In addition to video distribution, the Internet is also being used to transport phone calls. In fact, over the next 10 years, the Internet, along with wireless Internet access, may make the traditional circuit-switched telephone system a thing of the past. VoIP not only provides phone service inexpensively, but also provides numerous value-added services, such as video conferencing, online directory services, voice messaging, and integration into social networks such as Facebook and WeChat.

In Section 9.1, we described the intrinsic characteristics of video and voice, and then classified multimedia applications into three categories: (i) streaming stored audio/video, (ii) conversational voice/video-over-IP, and (iii) streaming live audio/video.

In Section 9.2, we studied streaming stored video in some depth. For streaming video applications, prerecorded videos are placed on servers, and users send requests to these servers to view the videos on demand. We saw that streaming video systems can be classified into two categories: UDP streaming and HTTP. We observed that the most important performance measure for streaming video is average throughput.

In Section 9.3, we examined how conversational multimedia applications, such as VoIP, can be designed to run over a best-effort network. For conversational multimedia, timing considerations are important because conversational applications are highly delay-sensitive. On the other hand, conversational multimedia applications are loss—tolerant—occasional loss only causes occasional glitches in audio/video playback, and these losses can often be partially or fully concealed. We saw how a combination of client buffers, packet sequence numbers, and timestamps can greatly alleviate the effects of network-induced jitter. We also surveyed the technology behind Skype, one of the leading voice- and video-over-IP companies. In Section 9.4, we examined two of the most important standardized protocols for VoIP, namely, RTP and SIP.

In Section 9.5, we introduced how several network mechanisms (link-level scheduling disciplines and traffic policing) can be used to provide differentiated service among several classes of traffic.

Homework Problems and Questions

Chapter 9 Review Questions

SECTION 9.1

R1. Reconstruct Table 9.1 for when Victor Video is watching a 4 Mbps video, Facebook Frank is looking at a new 100 Kbyte image every 20 seconds, and Martha Music is listening to 200 kbps audio stream.

R2. There are two types of redundancy in video. Describe them, and discuss how they can be exploited for efficient compression.

R3. Suppose an analog audio signal is sampled 16,000 times per second, and each sample is quantized into one of 1024 levels. What would be the resulting bit rate of the PCM digital audio signal?

R4. Multimedia applications can be classified into three categories. Name and describe each category.

SECTION 9.2

R5. Streaming video systems can be classified into three categories. Name and briefly describe each of these categories.

R6. List three disadvantages of UDP streaming.

R7. With HTTP streaming, are the TCP receive buffer and the client's application buffer the same thing? If not, how do they interact?

R8. Consider the simple model for HTTP streaming. Suppose the server sends bits at a constant rate of 2 Mbps and playback begins when 8 million bits have been received. What is the initial buffering delay t_p?

SECTION 9.3

R9. What is the difference between end-to-end delay and packet jitter? What are the causes of packet jitter?

R10. Why is a packet that is received after its scheduled playout time considered lost?

R11. Section 9.3 describes two FEC schemes. Briefly summarize them. Both schemes increase the transmission rate of the stream by adding overhead. Does interleaving also increase the transmission rate?

SECTION 9.4

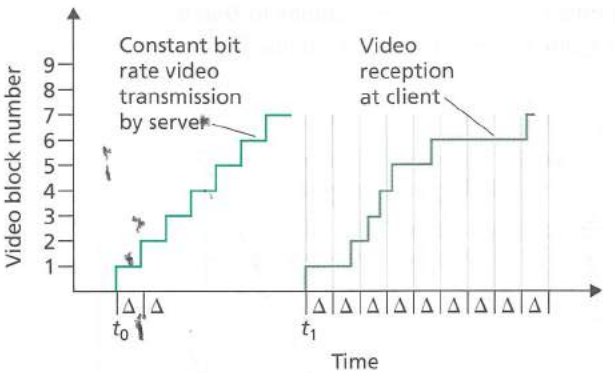
R12. How are different RTP streams in different sessions identified by a receiver? How are different streams from within the same session identified?

R13. What is the role of a SIP registrar? How is the role of an SIP registrar different from that of a home agent in Mobile IP?

Problems

P1. Consider the figure below. Similar to our discussion of Figure 9.1, suppose that video is encoded at a fixed bit rate, and thus each video block contains video frames that are to be played out over the same fixed amount of time, Δ. The server transmits the first video block at t_0, the second block at $t_0 + \Delta$,

the third block at $t_0 + 2\Delta$, and so on. Once the client begins playout, each block should be played out Δ time units after the previous block.



a. Suppose that the client begins playout as soon as the first block arrives at t_1. In the figure below, how many blocks of video (including the first block) will have arrived at the client in time for their playout? Explain how you arrived at your answer.

b. Suppose that the client begins playout now at $t_1 + \Delta$. How many blocks of video (including the first block) will have arrived at the client in time for their playout? Explain how you arrived at your answer.

c. In the same scenario at (b) above, what is the largest number of blocks that is ever stored in the client buffer, awaiting playout? Explain how you arrived at your answer.

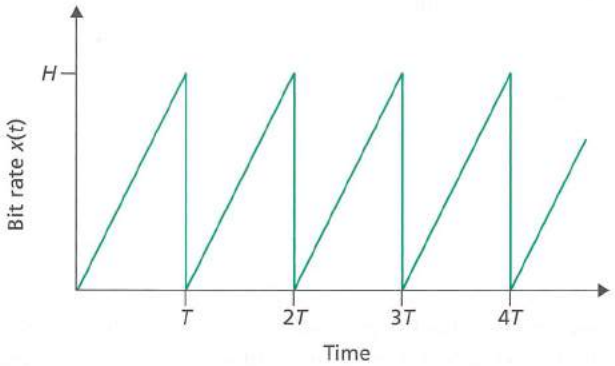
d. What is the smallest playout delay at the client, such that every video block has arrived in time for its playout? Explain how you arrived at your answer.

P2. Recall the simple model for HTTP streaming shown in Figure 9.3. Recall that B denotes the size of the client's application buffer, and Q denotes the number of bits that must be buffered before the client application begins playout. Also r denotes the video consumption rate. Assume that the server sends bits at a constant rate x whenever the client buffer is not full.

a. Suppose that $x < r$. As discussed in the text, in this case playout will alternate between periods of continuous playout and periods of freezing. Determine the length of each continuous playout and freezing period as a function of Q, r, and x.

b. Now suppose that $x > r$. At what time $t = t_f$ does the client application buffer become full?

P3. Recall the simple model for HTTP streaming shown in Figure 9.3. Suppose the buffer size is infinite but the server sends bits at variable rate $x(t)$. Specifically, suppose $x(t)$ has the following saw-tooth shape. The rate is initially zero at time $t = 0$ and linearly climbs to H at time $t = T$. It then repeats this pattern again and again, as shown in the figure below.



a. What is the server's average send rate?

b. Suppose that $Q = 0$, so that the client starts playback as soon as it receives a video frame. What will happen?

c. Now suppose $Q > 0$ and $HT/2 \geq Q$. Determine as a function of Q, H, and T the time at which playback first begins.

d. Suppose $H > 2r$ and $Q = HT/2$. Prove there will be no freezing after the initial playout delay.

e. Suppose $H > 2r$. Find the smallest value of Q such that there will be no freezing after the initial playback delay.

f. Now suppose that the buffer size B is finite. Suppose $H > 2r$. As a function of Q, B, T, and H, determine the time $t = t_f$ when the client application buffer first becomes full.

P4. Recall the simple model for HTTP streaming shown in Figure 9.3. Suppose the client application buffer is infinite, the server sends at the constant rate x, and the video consumption rate is r with $r < x$. Also suppose playback begins immediately. Suppose that the user terminates the video early at time $t = E$. At the time of termination, the server stops sending bits (if it hasn't already sent all the bits in the video).

a. Suppose the video is infinitely long. How many bits are wasted (that is, sent but not viewed)?

b. Suppose the video is T seconds long with $T > E$. How many bits are wasted (that is, sent but not viewed)?

P5. Consider a DASH system (as discussed in Section 2.6) for which there are N video versions (at N different rates and qualities) and N audio versions (at N different rates and qualities). Suppose we want to allow the player to choose at any time any of the N video versions and any of the N audio versions.

a. If we create files so that the audio is mixed in with the video, so server sends only one media stream at given time, how many files will the server need to store (each a different URL)?

b. If the server instead sends the audio and video streams separately and has the client synchronize the streams, how many files will the server need to store?

P6. In the VoIP example in Section 9.3, let h be the total number of header bytes added to each chunk, including UDP and IP header.

a. Assuming an IP datagram is emitted every 20 msecs, find the transmission rate in bits per second for the datagrams generated by one side of this application.

b. What is a typical value of h when RTP is used?

P7. Consider the procedure described in Section 9.3 for estimating average delay d_i. Suppose that $u = 0.1$. Let $r_1 - t_1$ be the most recent sample delay, let $r_2 - t_2$ be the next most recent sample delay, and so on.

a. For a given audio application suppose four packets have arrived at the receiver with sample delays $r_4 - t_4$, $r_3 - t_3$, $r_2 - t_2$, and $r_1 - t_1$. Express the estimate of delay d in terms of the four samples.

b. Generalize your formula for n sample delays.

c. For the formula in part (b), let n approach infinity and give the resulting formula. Comment on why this averaging procedure is called an exponential moving average.

P8. Repeat parts (a) and (b) in Question P7 for the estimate of average delay deviation.

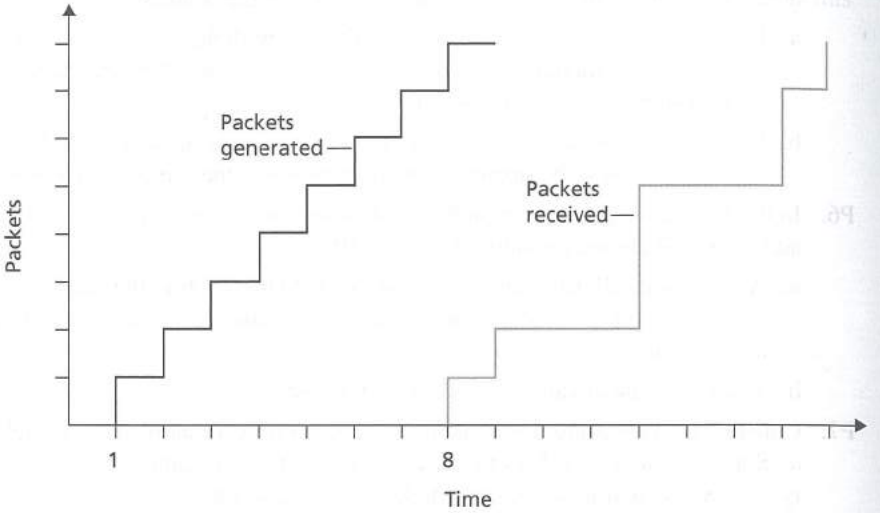
P9. For the VoIP example in Section 9.3, we introduced an online procedure (exponential moving average) for estimating delay. In this problem we will examine an alternative procedure. Let t_i be the timestamp of the ith packet received; let r_i be the time at which the ith packet is received. Let d_n be our estimate of average delay after receiving the nth packet. After the first packet is received, we set the delay estimate equal to $d_1 = r_1 - t_1$.

a. Suppose that we would like $d_n = (r_1 - t_1 + r_2 - t_2 + \cdots + r_n - t_n)/n$ for all n. Give a recursive formula for d_n in terms of d_{n-1}, r_n, and t_n.

b. Describe why for Internet telephony, the delay estimate described in Section 9.3 is more appropriate than the delay estimate outlined in part (a).

P10. Compare the procedure described in Section 9.3 for estimating average delay with the procedure in Section 3.5 for estimating round-trip time. What do the procedures have in common? How are they different?

P11. Consider the figure below (which is similar to Figure 9.3). A sender begins sending packetized audio periodically at $t = 1$. The first packet arrives at the receiver at $t = 8$.



a. What are the delays (from sender to receiver, ignoring any playout delays) of packets 2 through 8? Note that each vertical and horizontal line segment in the figure has a length of 1, 2, or 3 time units.

b. If audio playout begins as soon as the first packet arrives at the receiver at $t = 8$, which of the first eight packets sent will *not* arrive in time for playout?

c. If audio playout begins at $t = 9$, which of the first eight packets sent will not arrive in time for playout?

d. What is the minimum playout delay at the receiver that results in all of the first eight packets arriving in time for their playout?

P12. Consider again the figure in P11, showing packet audio transmission and reception times.

a. Compute the estimated delay for packets 2 through 8, using the formula for d_i from Section 9.3.2. Use a value of $u = 0.1$.

b. Compute the estimated deviation of the delay from the estimated average for packets 2 through 8, using the formula for v_i from Section 9.3.2. Use a value of $u = 0.1$.

P13. Recall the two FEC schemes for VoIP described in Section 9.3. Suppose the first scheme generates a redundant chunk for every four original chunks. Suppose the second scheme uses a low-bit rate encoding whose transmission rate is 25 percent of the transmission rate of the nominal stream.

a. How much additional bandwidth does each scheme require? How much playback delay does each scheme add?

b. How do the two schemes perform if the first packet is lost in every group of five packets? Which scheme will have better audio quality?

c. How do the two schemes perform if the first packet is lost in every group of two packets? Which scheme will have better audio quality?

P14. a. Consider an audio conference call in Skype with $N > 2$ participants. Suppose each participant generates a constant stream of rate r bps. How many bits per second will the call initiator need to send? How many bits per second will each of the other $N - 1$ participants need to send? What is the total send rate, aggregated over all participants?

b. Repeat part (a) for a Skype video conference call using a central server.

c. Repeat part (b), but now for when each peer sends a copy of its video stream to each of the $N - 1$ other peers.

P15. a. Suppose we send into the Internet two IP datagrams, each carrying a different UDP segment. The first datagram has source IP address A1, destination IP address B, source port P1, and destination port T. The second datagram has source IP address A2, destination IP address B, source port P2, and destination port T. Suppose that A1 is different from A2 and that P1 is different from P2. Assuming that both datagrams reach their final destination, will the two UDP datagrams be received by the same socket? Why or why not?

b. Suppose Alice, Bob, and Claire want to have an audio conference call using SIP and RTP. For Alice to send and receive RTP packets to and from Bob and Claire, is only one UDP socket sufficient (in addition to the socket needed for the SIP messages)? If yes, then how does Alice's SIP client distinguish between the RTP packets received from Bob and Claire?

P16. True or false:

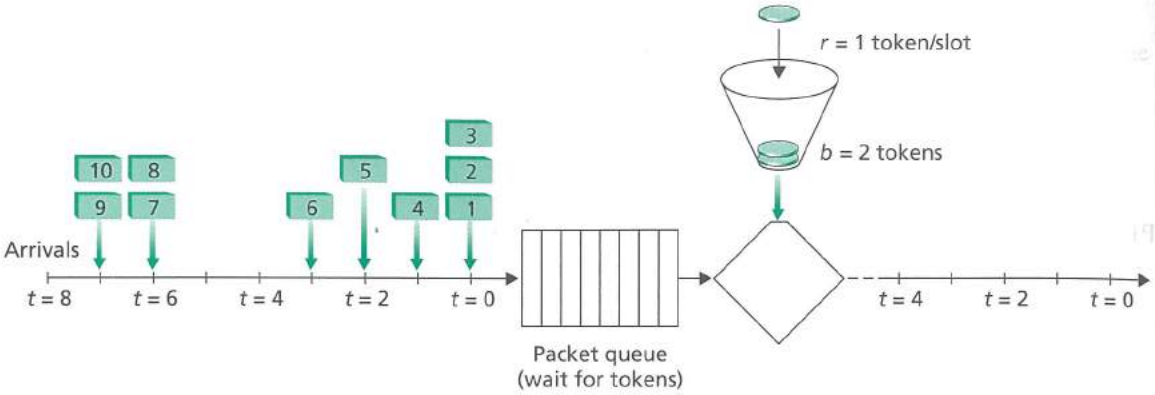
a. If stored video is streamed directly from a Web server to a media player, then the application is using TCP as the underlying transport protocol.

b. When using RTP, it is possible for a sender to change encoding in the middle of a session.

c. All applications that use RTP must use port 87.

d. If an RTP session has a separate audio and video stream for each sender, then the audio and video streams use the same SSRC.

e. In differentiated services, while per-hop behavior defines differences in performance among classes, it does not mandate any particular mechanism for achieving these performances.



f. Suppose Alice wants to establish an SIP session with Bob. In her INVITE message she includes the line: m=audio 48753 RTP/AVP 3 (AVP 3 denotes GSM audio). Alice has therefore indicated in this message that she wishes to send GSM audio.

g. Referring to the preceding statement, Alice has indicated in her INVITE message that she will send audio to port 48753.

h. SIP messages are typically sent between SIP entities using a default SIP port number.

i. In order to maintain registration, SIP clients must periodically send REGISTER messages.

j. SIP mandates that all SIP clients support G.711 audio encoding.

P17. Consider the figure below, which shows a leaky bucket policer being fed by a stream of packets. The token buffer can hold at most two tokens, and is initially full at $t = 0$. New tokens arrive at a rate of one token per slot. The output link speed is such that if two packets obtain tokens at the beginning of a time slot, they can both go to the output link in the same slot. The timing details of the system are as follows:

1. Packets (if any) arrive at the beginning of the slot. Thus in the figure, packets 1, 2, and 3 arrive in slot 0. If there are already packets in the queue, then the arriving packets join the end of the queue. Packets proceed towards the front of the queue in a FIFO manner.

2. After the arrivals have been added to the queue, if there are any queued packets, one or two of those packets (depending on the number of available tokens) will each remove a token from the token buffer and go to the output link during that slot. Thus, packets 1 and 2 each remove a token from the buffer (since there are initially two tokens) and go to the output link during slot 0.

3. A new token is added to the token buffer if it is not full, since the token generation rate is $r = 1$ token/slot.
4. Time then advances to the next time slot, and these steps repeat.

Answer the following questions:

a. For each time slot, identify the packets that are in the queue and the number of tokens in the bucket, immediately after the arrivals have been processed (step 1 above) but before any of the packets have passed through the queue and removed a token. Thus, for the $t = 0$ time slot in the example above, packets 1, 2, and 3 are in the queue, and there are two tokens in the buffer.

b. For each time slot indicate which packets appear on the output after the token(s) have been removed from the queue. Thus, for the $t = 0$ time slot in the example above, packets 1 and 2 appear on the output link from the leaky buffer during slot 0.

P18. Repeat P17 but assume that $r = 2$. Assume again that the bucket is initially full.

P19. Consider P18 and suppose now that $r = 3$ and that $b = 2$ as before. Will your answer to the question above change?

P20. Consider the leaky bucket policer that polices the average rate and burst size of a packet flow. We now want to police the peak rate, p, as well. Show how the output of this leaky bucket policer can be fed into a second leaky bucket policer so that the two leaky buckets in series police the average rate, peak rate, and burst size. Be sure to give the bucket size and token generation rate for the second policer.

P21. A packet flow is said to conform to a leaky bucket specification (r, b) with burst size b and average rate r if the number of packets that arrive to the leaky bucket is less than $rt + b$ packets in every interval of time of length t for all t. Will a packet flow that conforms to a leaky bucket specification (r, b) ever have to wait at a leaky bucket policer with parameters r and b? Justify your answer.

P22. Show that as long as $r_1 < Rw_1/(\sum w_j)$, then d_{max} is indeed the maximum delay that any packet in flow 1 will ever experience in the WFQ queue.

Programming Assignment

In this lab, you will implement a streaming video server and client. The client will use the real-time streaming protocol (RTSP) to control the actions of the server. The server will use the real-time protocol (RTP) to packetize the video for transport over UDP. You will be given Python code that partially implements RTSP and RTP at the client and server. Your job will be to complete both the client and server code.

When you are finished, you will have created a client-server application that does the following:

- The client sends SETUP, PLAY, PAUSE, and TEARDOWN RTSP commands, and the server responds to the commands.
- When the server is in the playing state, it periodically grabs a stored JPEG frame, packetizes the frame with RTP, and sends the RTP packet into a UDP socket.
- The client receives the RTP packets, removes the JPEG frames, decompresses the frames, and renders the frames on the client's monitor.

The code you will be given implements the RTSP protocol in the server and the RTP depacketization in the client. The code also takes care of displaying the transmitted video. You will need to implement RTSP in the client and RTP server. This programming assignment will significantly enhance the student's understanding of RTP, RTSP, and streaming video. It is highly recommended. The assignment also suggests a number of optional exercises, including implementing the RTSP DESCRIBE command at both client and server. You can find full details of the assignment, as well as an overview of the RTSP protocol, at the Web site www.pearsonhighered.com/cs-resources.

Henning Schulzrinne

Henning Schulzrinne is a professor, chair of the Department of Computer Science, and head of the Internet Real-Time Laboratory at Columbia University. He is the co-author of RTP, RTSP, SIP, and GIST—key protocols for audio and video communications over the Internet. Henning received his BS in electrical and industrial engineering at TU Darmstadt in Germany, his MS in electrical and computer engineering at the University of Cincinnati, and his PhD in electrical engineering at the University of Massachusetts, Amherst.

What made you decide to specialize in multimedia networking?

This happened almost by accident. As a PhD student, I got involved with DARTnet, an experimental network spanning the United States with T1 lines. DARTnet was used as a proving ground for multicast and Internet real-time tools. That led me to write my first audio tool, NeVoT. Through some of the DARTnet participants, I became involved in the IETF, in the then-nascent Audio Video Transport working group. This group later ended up standardizing RTP.

What was your first job in the computer industry? What did it entail?

My first job in the computer industry was soldering together an Altair computer kit when I was a high school student in Livermore, California. Back in Germany, I started a little consulting company that devised an address management program for a travel agency—storing data on cassette tapes for our TRS-80 and using an IBM Selectric typewriter with a home-brew hardware interface as a printer.

My first real job was with AT&T Bell Laboratories, developing a network emulator for constructing experimental networks in a lab environment.

What are the goals of the Internet Real-Time Lab?

Our goal is to provide components and building blocks for the Internet as the single future communications infrastructure. This includes developing new protocols, such as GIST (for network-layer signaling) and LoST (for finding resources by location), or enhancing protocols that we have worked on earlier, such as SIP, through work on rich presence, peer-to-peer systems, next-generation emergency calling, and service creation tools. Recently, we have also looked extensively at wireless systems for VoIP, as 802.11b and 802.11n networks and maybe WiMax networks are likely to become important last-mile technologies for telephony. We are also trying to greatly improve the ability of users to diagnose faults in the complicated tangle of providers and equipment, using a peer-to-peer fault diagnosis system called DYSWIS (Do You See What I See).

We try to do practically relevant work, by building prototypes and open source systems, by measuring performance of real systems, and by contributing to IETF standards.

What is your vision for the future of multimedia networking?

We are now in a transition phase; just a few years shy of when IP will be the universal platform for multimedia services, from IPTV to VoIP. We expect radio, telephone, and TV to be available even during snowstorms and earthquakes, so when the Internet takes over the role of these dedicated networks, users will expect the same level of reliability.

We will have to learn to design network technologies for an ecosystem of competing carriers, service and content providers, serving lots of technically untrained users and defending them against a small, but destructive, set of malicious and criminal users. Changing protocols is becoming increasingly hard. They are also becoming more complex, as they need to take into account competing business interests, security, privacy, and the lack of transparency of networks caused by firewalls and network address translators.

Since multimedia networking is becoming the foundation for almost all of consumer entertainment, there will be an emphasis on managing very large networks, at low cost. Users will expect ease of use, such as finding the same content on all of their devices.

Why does SIP have a promising future?

As the current wireless network upgrade to 3G networks proceeds, there is the hope of a single multimedia signaling mechanism spanning all types of networks, from cable modems, to corporate telephone networks and public wireless networks. Together with software radios, this will make it possible in the future that a single device can be used on a home network, as a cordless BlueTooth phone, in a corporate network via 802.11 and in the wide area via 3G networks. Even before we have such a single universal wireless device, the personal mobility mechanisms make it possible to hide the differences between networks. One identifier becomes the universal means of reaching a person, rather than remembering or passing around half a dozen technology- or location-specific telephone numbers.

SIP also breaks apart the provision of voice (bit) transport from voice services. It now becomes technically possible to break apart the local telephone monopoly, where one company provides neutral bit transport, while others provide IP "dial tone" and the classical telephone services, such as gateways, call forwarding, and caller ID.

Beyond multimedia signaling, SIP offers a new service that has been missing in the Internet: event notification. We have approximated such services with HTTP kludges and e-mail, but this was never very satisfactory. Since events are a common abstraction for distributed systems, this may simplify the construction of new services.

Do you have any advice for students entering the networking field?

Networking bridges disciplines. It draws from electrical engineering, all aspects of computer science, operations research, statistics, economics, and other disciplines. Thus, networking researchers have to be familiar with subjects well beyond protocols and routing algorithms. Given that networks are becoming such an important part of everyday life, students wanting to make a difference in the field should think of the new resource constraints in networks: human time and effort, rather than just bandwidth or storage.

Work in networking research can be immensely satisfying since it is about allowing people to communicate and exchange ideas, one of the essentials of being human. The Internet has become the third major global infrastructure, next to the transportation system and energy distribution. Almost no part of the economy can work without high-performance networks, so there should be plenty of opportunities for the foreseeable future.

References

A note on URLs. In the references below, we have provided URLs for Web pages, Web-only documents, and other material that has not been published in a conference or journal (when we have been able to locate a URL for such material). We have not provided URLs for conference and journal publications, as these documents can usually be located via a search engine, from the conference Web site (e.g., papers in all *ACM SIGCOMM* conferences and workshops can be located via http://www.acm.org/sigcomm), or via a digital library subscription. While all URLs provided below were valid (and tested) in Jan. 2016, URLs can become out of date. Please consult the online version of this book (www.pearsonhighered .com/cs-resources) for an up-to-date bibliography.

A note on Internet Request for Comments (RFCs): Copies of Internet RFCs are available at many sites. The RFC Editor of the Internet Society (the body that oversees the RFCs) maintains the site, http://www.rfc-editor.org. This site allows you to search for a specific RFC by title, number, or authors, and will show updates to any RFCs listed. Internet RFCs can be updated or obsoleted by later RFCs. Our favorite site for getting RFCs is the original source—http://www.rfc-editor.org.

[3GPP 2016] Third Generation Partnership Project homepage, http://www.3gpp.org/

[Abramson 1970] N. Abramson, "The Aloha System—Another Alternative for Computer Communications," *Proc. 1970 Fall Joint Computer Conference, AFIPS Conference*, p. 37, 1970.

[Abramson 1985] N. Abramson, "Development of the Alohanet," *IEEE Transactions on Information Theory*, Vol. IT-31, No. 3 (Mar. 1985), pp. 119–123.

[Abramson 2009] N. Abramson, "The Alohanet—Surfing for Wireless Data," *IEEE Communications Magazine*, Vol. 47, No. 12, pp. 21–25.

[Adhikari 2011a] V. K. Adhikari, S. Jain, Y. Chen, Z. L. Zhang, "Vivisecting YouTube: An Active Measurement Study," Technical Report, University of Minnesota, 2011.

[Adhikari 2012] V. K. Adhikari, Y. Gao, F. Hao, M. Varvello, V. Hilt, M. Steiner, Z. L. Zhang, "Unreeling Netflix: Understanding and Improving Multi-CDN Movie Delivery," Technical Report, University of Minnesota, 2012.

[Afanasyev 2010] A. Afanasyev, N. Tilley, P. Reiher, L. Kleinrock, "Host-to-Host Congestion Control for TCP," *IEEE Communications Surveys & Tutorials*, Vol. 12, No. 3, pp. 304–342.

[Agarwal 2009] S. Agarwal, J. Lorch, "Matchmaking for Online Games and Other Latency-sensitive P2P Systems," *Proc. 2009 ACM SIGCOMM.*

[Ager 2012] B. Ager, N. Chatzis, A. Feldmann, N. Sarrar, S. Uhlig, W. Willinger, "Anatomy of a Large European ISP," Sigcomm, 2012.

[Ahn 1995] J. S. Ahn, P. B. Danzig, Z. Liu, and Y. Yan, "Experience with TCP Vegas: Emulation and Experiment," *Proc. 1995 ACM SIGCOMM* (Boston, MA, Aug. 1995), pp. 185–195.

[Akamai 2016] Akamai homepage, http://www.akamai.com

[Akella 2003] A. Akella, S. Seshan, A. Shaikh, "An Empirical Evaluation of Wide-Area Internet Bottlenecks," *Proc. 2003 ACM Internet Measurement Conference* (Miami, FL, Nov. 2003).

[Akhshabi 2011] S. Akhshabi, A. C. Begen, C. Dovrolis, "An Experimental Evaluation of Rate-Adaptation Algorithms in Adaptive Streaming over HTTP," *Proc. 2011 ACM Multimedia Systems Conf.*

[Akyildiz 2010] I. Akyildiz, D. Gutierrex-Estevez, E. Reyes, "The Evolution to 4G Cellular Systems, LTE Advanced," *Physical Communication*, Elsevier, 3 (2010), 217–244.

[Albitz 1993] P. Albitz and C. Liu, *DNS and BIND*, O'Reilly & Associates, Petaluma, CA, 1993.

[Al-Fares 2008] M. Al-Fares, A. Loukissas, A. Vahdat, "A Scalable, Commodity Data Center Network Architecture," *Proc. 2008 ACM SIGCOMM.*

[Amazon 2014] J. Hamilton, *"AWS: Innovation at Scale*, YouTube video, https://www.youtube.com/watch?v=JIQETrFC_SQ

[Anderson 1995] J. B. Andersen, T. S. Rappaport, S. Yoshida, "Propagation Measurements and Models for Wireless Communications Channels," *IEEE Communications Magazine*, (Jan. 1995), pp. 42–49.

[Alizadeh 2010] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan. "Data center TCP (DCTCP)," *ACM SIGCOMM 2010 Conference*, ACM, New York, NY, USA, pp. 63–74.

[Allman 2011] E. Allman, "The Robustness Principle Reconsidered: Seeking a Middle Ground," *Communications of the ACM*, Vol. 54, No. 8 (Aug. 2011), pp. 40–45.

[Appenzeller 2004] G. Appenzeller, I. Keslassy, N. McKeown, "Sizing Router Buffers," *Proc. 2004 ACM SIGCOMM* (Portland, OR, Aug. 2004).

[ASO-ICANN 2016] The Address Supporting Organization homepage, http://www.aso.icann.org

[AT&T 2013] "AT&T Vision Alignment Challenge Technology Survey," AT&T Domain 2.0 Vision White Paper, November 13, 2013.

[Atheros 2016] Atheros Communications Inc., "Atheros AR5006 WLAN Chipset Product Bulletins," http://www.atheros.com/pt/AR5006Bulletins.htm

[Ayanoglu 1995] E. Ayanoglu, S. Paul, T. F. La Porta, K. K. Sabnani, R. D. Gitlin, "AIRMAIL: A Link-Layer Protocol for Wireless Networks," *ACM ACM/Baltzer Wireless Networks Journal*, 1: 47–60, Feb. 1995.

[Bakre 1995] A. Bakre, B. R. Badrinath, "I-TCP: Indirect TCP for Mobile Hosts," *Proc. 1995 Int. Conf. on Distributed Computing Systems (ICDCS)* (May 1995), pp. 136–143.

[Balakrishnan 1997] H. Balakrishnan, V. Padmanabhan, S. Seshan, R. Katz, "A Comparison of Mechanisms for Improving TCP Performance Over Wireless Links," *IEEE/ACM Transactions on Networking* Vol. 5, No. 6 (Dec. 1997).

[Balakrishnan 2003] H. Balakrishnan, F. Kaashoek, D. Karger, R. Morris, I. Stoica, "Looking Up Data in P2P Systems," *Communications of the ACM*, Vol. 46, No. 2 (Feb. 2003), pp. 43–48.

[Baldauf 2007] M. Baldauf, S. Dustdar, F. Rosenberg, "A Survey on Context-Aware Systems," *Int. J. Ad Hoc and Ubiquitous Computing*, Vol. 2, No. 4 (2007), pp. 263–277.

[Baran 1964] P. Baran, "On Distributed Communication Networks," *IEEE Transactions on Communication Systems*, Mar. 1964. Rand Corporation Technical report with the same title (Memorandum RM-3420-PR, 1964). http://www.rand.org/publications/RM/RM3420/

[Bardwell 2004] J. Bardwell, "You Believe You Understand What You Think I Said . . . The Truth About 802.11 Signal and Noise Metrics: A Discussion Clarifying Often-Misused 802.11 WLAN Terminologies," http://www.connect802.com/download/techpubs/2004/you_believe_D100201.pdf

[Barford 2009] P. Barford, N. Duffield, A. Ron, J. Sommers, "Network Performance Anomaly Detection and Localization," *Proc. 2009 IEEE INFOCOM* (Apr. 2009).

[Baronti 2007] P. Baronti, P. Pillai, V. Chook, S. Chessa, A. Gotta, Y. Hu, "Wireless Sensor Networks: A Survey on the State of the Art and the 802.15.4 and ZigBee Standards," *Computer Communications*, Vol. 30, No. 7 (2007), pp. 1655–1695.

[Baset 2006] S. A. Basset and H. Schulzrinne, "An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol," *Proc. 2006 IEEE INFOCOM* (Barcelona, Spain, Apr. 2006).

[BBC 2001] BBC news online "A Small Slice of Design," Apr. 2001, http://news.bbc.co.uk/2/hi/science/nature/1264205.stm

[Beheshti 2008] N. Beheshti, Y. Ganjali, M. Ghobadi, N. McKeown, G. Salmon, "Experimental Study of Router Buffer Sizing," *Proc. ACM Internet Measurement Conference* (Oct. 2008, Vouliagmeni, Greece).

[Bender 2000] P. Bender, P. Black, M. Grob, R. Padovani, N. Sindhushayana, A. Viterbi, "CDMA/HDR: A Bandwidth-Efficient High-Speed Wireless Data Service for Nomadic Users," *IEEE Commun. Mag.*, Vol. 38, No. 7 (July 2000), pp. 70–77.

[Berners-Lee 1989] T. Berners-Lee, CERN, "Information Management: A Proposal," Mar. 1989, May 1990. http://www.w3.org/History/1989/proposal.html

[Berners-Lee 1994] T. Berners-Lee, R. Cailliau, A. Luotonen, H. Frystyk Nielsen, A. Secret, "The World-Wide Web," *Communications of the ACM*, Vol. 37, No. 8 (Aug. 1994), pp. 76–82.

[Bertsekas 1991] D. Bertsekas, R. Gallagher, *Data Networks, 2nd Ed.*, Prentice Hall, Englewood Cliffs, NJ, 1991.

[Biersack 1992] E. W. Biersack, "Performance Evaluation of Forward Error Correction in ATM Networks," *Proc. 1999 ACM SIGCOMM* (Baltimore, MD, Aug. 1992), pp. 248–257.

[BIND 2016] Internet Software Consortium page on BIND, http://www.isc.org/bind.html

[Bisdikian 2001] C. Bisdikian, "An Overview of the Bluetooth Wireless Technology," *IEEE Communications Magazine*, No. 12 (Dec. 2001), pp. 86–94.

[Bishop 2003] M. Bishop, *Computer Security: Art and Science*, Boston: Addison Wesley, Boston MA, 2003.

[Black 1995] U. Black, *ATM Volume I: Foundation for Broadband Networks*, Prentice Hall, 1995.

[Black 1997] U. Black, *ATM Volume II: Signaling in Broadband Networks*, Prentice Hall, 1997.

[Blumenthal 2001] M. Blumenthal, D. Clark, "Rethinking the Design of the Internet: The End-to-end Arguments vs. the Brave New World," *ACM Transactions on Internet Technology*, Vol. 1, No. 1 (Aug. 2001), pp. 70–109.

[Bochman 1984] G. V. Bochmann, C. A. Sunshine, "Formal Methods in Communication Protocol Design," *IEEE Transactions on Communications*, Vol. 28, No. 4 (Apr. 1980) pp. 624–631.

[Bolot 1996] J-C. Bolot, A. Vega-Garcia, "Control Mechanisms for Packet Audio in the Internet," *Proc. 1996 IEEE INFOCOM*, pp. 232–239.

[Bosshart 2013] P. Bosshart, G. Gibb, H. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, M. Horowitz, "Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN," *ACM SIGCOMM Comput. Commun. Rev.* 43, 4 (Aug. 2013), 99–110.

[Bosshart 2014] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, D. Walker, "P4: Programming Protocol-Independent Packet Processors," *ACM SIGCOMM Comput. Commun. Rev.* 44, 3 (July 2014), pp. 87–95.

[Brakmo 1995] L. Brakmo, L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE Journal of Selected Areas in Communications*, Vol. 13, No. 8 (Oct. 1995), pp. 1465–1480.

[Bryant 1988] B. Bryant, "Designing an Authentication System: A Dialogue in Four Scenes," http://web.mit.edu/kerberos/www/dialogue.html

[Bush 1945] V. Bush, "As We May Think," *The Atlantic Monthly*, July 1945. http://www.theatlantic.com/unbound/flashbks/computer/bushf.htm

[Byers 1998] J. Byers, M. Luby, M. Mitzenmacher, A. Rege, "A Digital Fountain Approach to Reliable Distribution of Bulk Data," *Proc. 1998 ACM SIGCOMM* (Vancouver, Canada, Aug. 1998), pp. 56–67.

[Caesar 2005a] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, J. van der Merwe, "Design and implementation of a Routing Control Platform," *Proc. Networked Systems Design and Implementation* (May 2005).

[Caesar 2005b] M. Caesar, J. Rexford, "BGP Routing Policies in ISP Networks," *IEEE Network Magazine*, Vol. 19, No. 6 (Nov. 2005).

[Caldwell 2012] C. Caldwell, "The Prime Pages," http://www.utm.edu/research/primes/prove

[Cardwell 2000] N. Cardwell, S. Savage, T. Anderson, "Modeling TCP Latency," *Proc. 2000 IEEE INFOCOM* (Tel-Aviv, Israel, Mar. 2000).

[Casado 2007] M. Casado, M. Freedman, J. Pettit, J. Luo, N. McKeown, S. Shenker, "Ethane: Taking Control of the Enterprise," *Proc. ACM SIGCOMM '07*, New York, pp. 1–12. See also *IEEE/ACM Trans. Networking*, 17, 4 (Aug. 2007), pp. 270–1283.

[Casado 2009] M. Casado, M. Freedman, J. Pettit, J. Luo, N. Gude, N. McKeown, S. Shenker, "Rethinking Enterprise Network Control," *IEEE/ACM Transactions on Networking (ToN)*, Vol. 17, No. 4 (Aug. 2009), pp. 1270–1283.

[Casado 2014] M. Casado, N. Foster, A. Guha, "Abstractions for Software-Defined Networks," *Communications of the ACM*, Vol. 57 No. 10, (Oct. 2014), pp. 86–95.

[Cerf 1974] V. Cerf, R. Kahn, "A Protocol for Packet Network Interconnection," *IEEE Transactions on Communications Technology*, Vol. COM-22, No. 5, pp. 627–641.

[CERT 2001–09] CERT, "Advisory 2001–09: Statistical Weaknesses in TCP/IP Initial Sequence Numbers," http://www.cert.org/advisories/CA-2001-09.html

[CERT 2003–04] CERT, "CERT Advisory CA-2003-04 MS-SQL Server Worm," http://www.cert.org/advisories/CA-2003-04.html

[CERT 2016] CERT, http://www.cert.org

[CERT Filtering 2012] CERT, "Packet Filtering for Firewall Systems," http://www.cert.org/tech_tips/packet_filtering.html

[Cert SYN 1996] CERT, "Advisory CA-96.21: TCP SYN Flooding and IP Spoofing Attacks," http://www.cert.org/advisories/CA-1998-01.html

[Chandra 2007] T. Chandra, R. Greisemer, J. Redstone, "Paxos Made Live: an Engineering Perspective," Proc. of 2007 ACM Symposium on Principles of Distributed Computing (PODC), pp. 398–407.

[Chao 2001] H. J. Chao, C. Lam, E. Oki, *Broadband Packet Switching Technologies—A Practical Guide to ATM Switches and IP Routers*, John Wiley & Sons, 2001.

[Chao 2011] C. Zhang, P. Dunghel, D. Wu, K. W. Ross, "Unraveling the BitTorrent Ecosystem," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 22, No. 7 (July 2011).

[Chen 2000] G. Chen, D. Kotz, "A Survey of Context-Aware Mobile Computing Research," *Technical Report TR2000-381*, Dept. of Computer Science, Dartmouth College, Nov. 2000. http://www.cs.dartmouth.edu/reports/TR2000-381.pdf

[Chen 2006] K.-T. Chen, C.-Y. Huang, P. Huang, C.-L. Lei, "Quantifying Skype User Satisfaction," *Proc. 2006 ACM SIGCOMM* (Pisa, Italy, Sept. 2006).

[Chen 2011] Y. Chen, S. Jain, V. K. Adhikari, Z. Zhang, "Characterizing Roles of Front-End Servers in End-to-End Performance of Dynamic Content Distribution," *Proc. 2011 ACM Internet Measurement Conference* (Berlin, Germany, Nov. 2011).

[Cheswick 2000] B. Cheswick, H. Burch, S. Branigan, "Mapping and Visualizing the Internet," *Proc. 2000 Usenix Conference* (San Diego, CA, June 2000).

[Chiu 1989] D. Chiu, R. Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks," *Computer Networks and ISDN Systems*, Vol. 17, No. 1, pp. 1–14. http://www.cs.wustl.edu/~jain/papers/cong_av.htm

[Christiansen 2001] M. Christiansen, K. Jeffay, D. Ott, F. D. Smith, "Tuning Red for Web Traffic," *IEEE/ACM Transactions on Networking*, Vol. 9, No. 3 (June 2001), pp. 249–264.

[Chuang 2005] S. Chuang, S. Iyer, N. McKeown, "Practical Algorithms for Performance Guarantees in Buffered Crossbars," *Proc. 2005 IEEE INFOCOM.*

[Cisco 802.11ac 2014] Cisco Systems, "802.11ac: The Fifth Generation of Wi-Fi," Technical White Paper, Mar. 2014.

[Cisco 7600 2016] Cisco Systems, "Cisco 7600 Series Solution and Design Guide," http://www.cisco.com/en/US/products/hw/routers/ps368/prod_technical_reference09186a0080092246.html

[Cisco 8500 2012] Cisco Systems Inc., "Catalyst 8500 Campus Switch Router Architecture," http://www.cisco.com/univercd/cc/td/doc/product/l3sw/8540/rel_12_0/w5_6f/softcnfg/1cfg8500.pdf

[Cisco 12000 2016] Cisco Systems Inc., "Cisco XR 12000 Series and Cisco 12000 Series Routers," http://www.cisco.com/en/US/products/ps6342/index.html

[Cisco 2012] Cisco 2012, Data Centers, http://www.cisco.com/go/dce

[Cisco 2015] Cisco Visual Networking Index: Forecast and Methodology, 2014–2019, White Paper, 2015.

[Cisco 6500 2016] Cisco Systems, "Cisco Catalyst 6500 Architecture White Paper," http://www.cisco.com/c/en/us/products/collateral/switches/catalyst-6500-series-switches/prod_white_paper0900aecd80673385.html

[Cisco NAT 2016] Cisco Systems Inc., "How NAT Works," http://www.cisco.com/en/US/tech/tk648/tk361/technologies_tech_note09186a0080094831.shtml

[Cisco QoS 2016] Cisco Systems Inc., "Advanced QoS Services for the Intelligent Internet," http://www.cisco.com/warp/public/cc/pd/iosw/ioft/ioqo/tech/qos_wp.htm

[Cisco Queue 2016] Cisco Systems Inc., "Congestion Management Overview," http://www.cisco.com/en/US/docs/ios/12_2/qos/configuration/guide/qcfconmg.html

[Cisco SYN 2016] Cisco Systems Inc., "Defining Strategies to Protect Against TCP SYN Denial of Service Attacks," http://www.cisco.com/en/US/tech/tk828/technologies_tech_note09186a00800f67d5.shtml

[Cisco TCAM 2014] Cisco Systems Inc., "CAT 6500 and 7600 Series Routers and Switches TCAM Allocation Adjustment Procedures," http://www.cisco.com/c/en/us/support/docs/switches/catalyst-6500-series-switches/117712-problemsolution-cat6500-00.html

[Cisco VNI 2015] Cisco Systems Inc., "Visual Networking Index," http://www.cisco.com/web/solutions/sp/vni/vni_forecast_highlights/index.html

[Clark 1988] D. Clark, "The Design Philosophy of the DARPA Internet Protocols," *Proc. 1988 ACM SIGCOMM* (Stanford, CA, Aug. 1988).

[Cohen 1977] D. Cohen, "Issues in Transnet Packetized Voice Communication," *Proc. Fifth Data Communications Symposium* (Snowbird, UT, Sept. 1977), pp. 6–13.

[Cookie Central 2016] Cookie Central homepage, http://www.cookiecentral.com/n_cookie_faq.htm

[Cormen 2001] T. H. Cormen, *Introduction to Algorithms*, 2nd Ed., MIT Press, Cambridge, MA, 2001.

[Crow 1997] B. Crow, I. Widjaja, J. Kim, P. Sakai, "IEEE 802.11 Wireless Local Area Networks," *IEEE Communications Magazine* (Sept. 1997), pp. 116–126.

[Cusumano 1998] M. A. Cusumano, D. B. Yoffie, *Competing on Internet Time: Lessons from Netscape and Its Battle with Microsoft*, Free Press, New York, NY, 1998.

[Czyz 2014] J. Czyz, M. Allman, J. Zhang, S. Iekel-Johnson, E. Osterweil, M. Bailey, "Measuring IPv6 Adoption," *Proc. ACM SIGCOMM 2014*, ACM, New York, NY, USA, pp. 87–98.

[Dahlman 1998] E. Dahlman, B. Gudmundson, M. Nilsson, J. Sköld, "UMTS/IMT-2000 Based on Wideband CDMA," *IEEE Communications Magazine* (Sept. 1998), pp. 70–80.

[Daigle 1991] J. N. Daigle, *Queuing Theory for Telecommunications*, Addison-Wesley, Reading, MA, 1991.

[DAM 2016] Digital Attack Map, http://www.digitalattackmap.com

[Davie 2000] B. Davie and Y. Rekhter, *MPLS: Technology and Applications*, Morgan Kaufmann Series in Networking, 2000.

[Davies 2005] G. Davies, F. Kelly, "Network Dimensioning, Service Costing, and Pricing in a Packet-Switched Environment," *Telecommunications Policy*, Vol. 28, No. 4, pp. 391–412.

[DEC 1990] Digital Equipment Corporation, "In Memoriam: J. C. R. Licklider 1915–1990," SRC Research Report 61, Aug. 1990. http://www.memex.org/licklider.pdf

[DeClercq 2002] J. DeClercq, O. Paridaens, "Scalability Implications of Virtual Private Networks," *IEEE Communications Magazine*, Vol. 40, No. 5 (May 2002), pp. 151–157.

[Demers 1990] A. Demers, S. Keshav, S. Shenker, "Analysis and Simulation of a Fair Queuing Algorithm," *Internetworking: Research and Experience*, Vol. 1, No. 1 (1990), pp. 3–26.

[dhc 2016] IETF Dynamic Host Configuration working group homepage, http://www.ietf.org/html.charters/dhc-charter.html

[Dhungel 2012] P. Dhungel, K. W. Ross, M. Steiner., Y. Tian, X. Hei, "Xunlei: Peer-Assisted Download Acceleration on a Massive Scale," *Passive and Active Measurement Conference (PAM) 2012*, Vienna, 2012.

[Diffie 1976] W. Diffie, M. E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, Vol IT-22 (1976), pp. 644–654.

[Diggavi 2004] S. N. Diggavi, N. Al-Dhahir, A. Stamoulis, R. Calderbank, "Great Expectations: The Value of Spatial Diversity in Wireless Networks," *Proceedings of the IEEE*, Vol. 92, No. 2 (Feb. 2004).

[Dilley 2002] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, B. Weihl, "Globally Distributed Content Delivert," *IEEE Internet Computing* (Sept.–Oct. 2002).

[Diot 2000] C. Diot, B. N. Levine, B. Lyles, H. Kassem, D. Balensiefen, "Deployment Issues for the IP Multicast Service and Architecture," *IEEE Network*, Vol. 14, No. 1 (Jan./Feb. 2000) pp. 78–88.

[Dischinger 2007] M. Dischinger, A. Haeberlen, K. Gummadi, S. Saroiu, "Characterizing residential broadband networks," *Proc. 2007 ACM Internet Measurement Conference*, pp. 24–26.

[Dmitiropoulos 2007] X. Dmitiropoulos, D. Krioukov, M. Fomenkov, B. Huffaker, Y. Hyun, K. C. Claffy, G. Riley, "AS Relationships: Inference and Validation," *ACM Computer Communication Review* (Jan. 2007).

[DOCSIS 2011] Data-Over-Cable Service Interface Specifications, DOCSIS 3.0: MAC and Upper Layer Protocols Interface Specification, CM-SP-MULPIv3.0-I16-110623, 2011.

[Dodge 2016] M. Dodge, "An Atlas of Cyberspaces," http://www.cybergeography.org/atlas/isp_maps.html

[Donahoo 2001] M. Donahoo, K. Calvert, *TCP/IP Sockets in C: Practical Guide for Programmers*, Morgan Kaufman, 2001.

[DSL 2016] DSL Forum homepage, http://www.dslforum.org/

[Dhunghel 2008] P. Dhungel, D. Wu, B. Schonhorst, K.W. Ross, "A Measurement Study of Attacks on BitTorrent Leechers," *7th International Workshop on Peer-to-Peer Systems (IPTPS 2008)* (Tampa Bay, FL, Feb. 2008).

[Droms 2002] R. Droms, T. Lemon, *The DHCP Handbook* (2nd Edition), SAMS Publishing, 2002.

[Edney 2003] J. Edney and W. A. Arbaugh, *Real 802.11 Security: Wi-Fi Protected Access and 802.11i*, Addison-Wesley Professional, 2003.

[Edwards 2011] W. K. Edwards, R. Grinter, R. Mahajan, D. Wetherall, "Advancing the State of Home Networking," *Communications of the ACM*, Vol. 54, No. 6 (June 2011), pp. 62–71.

[Ellis 1987] H. Ellis, "The Story of Non-Secret Encryption," http://jya.com/ellis-doc.htm

[Erickson 2013] D. Erickson, " The Beacon Openflow Controller," 2nd *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking* (HotSDN '13). ACM, New York, NY, USA, pp. 13–18.

[Ericsson 2012] Ericsson, "The Evolution of Edge," http://www.ericsson.com/technology/whitepapers/broadband/evolution_of_EDGE.shtml

[Facebook 2014] A. Andreyev, "Introducing Data Center Fabric, the Next-Generation Facebook Data Center Network," https://code.facebook.com/posts/360346274145943/introducing-data-center-fabric-the-next-generation-facebook-data-center-network

[Faloutsos 1999] C. Faloutsos, M. Faloutsos, P. Faloutsos, "What Does the Internet Look Like? Empirical Laws of the Internet Topology," *Proc. 1999 ACM SIGCOMM* (Boston, MA, Aug. 1999).

[Farrington 2010] N. Farrington, G. Porter, S. Radhakrishnan, H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, A. Vahdat, "Helios: A Hybrid Electrical/Optical Switch Architecture for Modular Data Centers," *Proc. 2010 ACM SIGCOMM*.

[Feamster 2004] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, K. van der Merwe, "The Case for Separating Routing from Routers," *ACM SIGCOMM Workshop on Future Directions in Network Architecture*, Sept. 2004.

[Feamster 2004] N. Feamster, J. Winick, J. Rexford, "A Model for BGP Routing for Network Engineering," *Proc. 2004 ACM SIGMETRICS* (New York, NY, June 2004).

[Feamster 2005] N. Feamster, H. Balakrishnan, "Detecting BGP Configuration Faults with Static Analysis," *NSDI* (May 2005).

[Feamster 2013] N. Feamster, J. Rexford, E. Zegura, "The Road to SDN," *ACM Queue*, Volume 11, Issue 12, (Dec. 2013).

[Feldmeier 1995] D. Feldmeier, "Fast Software Implementation of Error Detection Codes," *IEEE/ACM Transactions on Networking*, Vol. 3, No. 6 (Dec. 1995), pp. 640–652.

[Ferguson 2013] A. Ferguson, A. Guha, C. Liang, R. Fonseca, S. Krishnamurthi, "Participatory Networking: An API for Application Control of SDNs," *Proceedings ACM SIGCOMM 2013*, pp. 327–338.

[Fielding 2000] R. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," 2000. PhD Thesis, UC Irvine, 2000.

[FIPS 1995] Federal Information Processing Standard, "Secure Hash Standard," FIPS Publication 180-1. http://www.itl.nist.gov/fipspubs/fip180-1.htm

[Floyd 1999] S. Floyd, K. Fall, "Promoting the Use of End-to-End Congestion Control in the Internet," *IEEE/ACM Transactions on Networking*, Vol. 6, No. 5 (Oct. 1998), pp. 458–472.

[Floyd 2000] S. Floyd, M. Handley, J. Padhye, J. Widmer, "Equation-Based Congestion Control for Unicast Applications," *Proc. 2000 ACM SIGCOMM* (Stockholm, Sweden, Aug. 2000).

[Floyd 2001] S. Floyd, "A Report on Some Recent Developments in TCP Congestion Control," *IEEE Communications Magazine* (Apr. 2001).

[Floyd 2016] S. Floyd, "References on RED (Random Early Detection) Queue Management," http://www.icir.org/floyd/red.html

[Floyd Synchronization 1994] S. Floyd, V. Jacobson, "Synchronization of Periodic Routing Messages," *IEEE/ACM Transactions on Networking*, Vol. 2, No. 2 (Apr. 1997) pp. 122–136.

[Floyd TCP 1994] S. Floyd, "TCP and Explicit Congestion Notification," *ACM SIGCOMM Computer Communications Review*, Vol. 24, No. 5 (Oct. 1994), pp. 10–23.

[Fluhrer 2001] S. Fluhrer, I. Mantin, A. Shamir, "Weaknesses in the Key Scheduling Algorithm of RC4," *Eighth Annual Workshop on Selected Areas in Cryptography* (Toronto, Canada, Aug. 2002).

[Fortz 2000] B. Fortz, M. Thorup, "Internet Traffic Engineering by Optimizing OSPF Weights," *Proc. 2000 IEEE INFOCOM* (Tel Aviv, Israel, Apr. 2000).

[Fortz 2002] B. Fortz, J. Rexford, M. Thorup, "Traffic Engineering with Traditional IP Routing Protocols," *IEEE Communication Magazine* (Oct. 2002).

[Fraleigh 2003] C. Fraleigh, F. Tobagi, C. Diot, "Provisioning IP Backbone Networks to Support Latency Sensitive Traffic," *Proc. 2003 IEEE INFOCOM* (San Francisco, CA, Mar. 2003).

[Frost 1994] J. Frost, "BSD Sockets: A Quick and Dirty Primer," http://world.std.com/~jimf/papers/sockets/sockets.html

[FTC 2015] Internet of Things: Privacy and Security in a Connected World, Federal Trade Commission, 2015, https://www.ftc.gov/system/files/documents/reports/federal-trade-commission-staff-report-november-2013-workshop-entitled-internet-things-privacy/150127iotrpt.pdf

[FTTH 2016] Fiber to the Home Council, http://www.ftthcouncil.org/

[Gao 2001] L. Gao, J. Rexford, "Stable Internet Routing Without Global Coordination," *IEEE/ACM Transactions on Networking*, Vol. 9, No. 6 (Dec. 2001), pp. 681–692.

[Gartner 2014] Gartner report on Internet of Things, http://www.gartner.com/technology/research/internet-of-things

[Gauthier 1999] L. Gauthier, C. Diot, and J. Kurose, "End-to-End Transmission Control Mechanisms for Multiparty Interactive Applications on the Internet," *Proc. 1999 IEEE INFOCOM* (New York, NY, Apr. 1999).

[Gember-Jacobson 2014] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, A. Akella, "OpenNF: Enabling Innovation in Network Function Control," *Proc. ACM SIGCOMM 2014*, pp. 163–174.

[Goodman 1997] David J. Goodman, *Wireless Personal Communications Systems*, Prentice-Hall, 1997.

[Google IPv6 2015] Google Inc. "IPv6 Statistics," https://www.google.com/intl/en/ipv6/statistics.html

[Google Locations 2016] Google data centers. http://www.google.com/corporate/datacenter/locations.html

[Goralski 1999] W. Goralski, *Frame Relay for High-Speed Networks*, John Wiley, New York, 1999.

[Greenberg 2009a] A. Greenberg, J. Hamilton, D. Maltz, P. Patel, "The Cost of a Cloud: Research Problems in Data Center Networks," *ACM Computer Communications Review* (Jan. 2009).

[Greenberg 2009b] A. Greenberg, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, S. Sengupta, "VL2: A Scalable and Flexible Data Center Network," *Proc. 2009 ACM SIGCOMM*.

[Greenberg 2011] A. Greenberg, J. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, S. Sengupta, "VL2: A Scalable and Flexible Data Center Network," *Communications of the ACM*, Vol. 54, No. 3 (Mar. 2011), pp. 95–104.

[Greenberg 2015] A. Greenberg, "SDN for the Cloud," Sigcomm 2015 Keynote Address, http://conferences.sigcomm.org/sigcomm/2015/pdf/papers/keynote.pdf

[Griffin 2012] T. Griffin, "Interdomain Routing Links," http://www.cl.cam.ac.uk/~tgg22/interdomain/

[Gude 2008] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: Towards an Operating System for Networks," *ACM SIGCOMM Computer Communication Review*, July 2008.

[Guha 2006] S. Guha, N. Daswani, R. Jain, "An Experimental Study of the Skype Peer-to-Peer VoIP System," *Proc. Fifth Int. Workshop on P2P Systems* (Santa Barbara, CA, 2006).

[Guo 2005] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, X. Zhang, "Measurement, Analysis, and Modeling of BitTorrent-Like Systems," *Proc. 2005 ACM Internet Measurement Conference*.

[Guo 2009] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, S. Lu, "BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers," *Proc. 2009 ACM SIGCOMM*.

[Gupta 2001] P. Gupta, N. McKeown, "Algorithms for Packet Classification," *IEEE Network Magazine*, Vol. 15, No. 2 (Mar./Apr. 2001), pp. 24–32.

[Gupta 2014] A. Gupta, L. Vanbever, M. Shahbaz, S. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, E. Katz-Bassett, "SDX: A Software Defined Internet Exchange, " *Proc. ACM SIGCOMM 2014* (Aug. 2014), pp. 551–562.

[Ha 2008] S. Ha, I. Rhee, L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," *ACM SIGOPS Operating System Review*, 2008.

[Halabi 2000] S. Halabi, *Internet Routing Architectures*, 2nd Ed., Cisco Press, 2000.

[Hanabali 2005] A. A. Hanbali, E. Altman, P. Nain, "A Survey of TCP over Ad Hoc Networks," *IEEE Commun. Surveys and Tutorials*, Vol. 7, No. 3 (2005), pp. 22–36.

[Hei 2007] X. Hei, C. Liang, J. Liang, Y. Liu, K. W. Ross, "A Measurement Study of a Large-scale P2P IPTV System," *IEEE Trans. on Multimedia* (Dec. 2007).

[Heidemann 1997] J. Heidemann, K. Obraczka, J. Touch, "Modeling the Performance of HTTP over Several Transport Protocols," *IEEE/ACM Transactions on Networking*, Vol. 5, No. 5 (Oct. 1997), pp. 616–630.

[Held 2001] G. Held, *Data Over Wireless Networks: Bluetooth, WAP, and Wireless LANs*, McGraw-Hill, 2001.

[Holland 2001] G. Holland, N. Vaidya, V. Bahl, "A Rate-Adaptive MAC Protocol for Multi-Hop Wireless Networks," *Proc. 2001 ACM Int. Conference of Mobile Computing and Networking (Mobicom01)* (Rome, Italy, July 2001).

[Hollot 2002] C.V. Hollot, V. Misra, D. Towsley, W. Gong, "Analysis and Design of Controllers for AQM Routers Supporting TCP Flows," *IEEE Transactions on Automatic Control*, Vol. 47, No. 6 (June 2002), pp. 945–959.

[Hong 2013] C. Hong, S, Kandula, R. Mahajan, M.Zhang, V. Gill, M. Nanduri, R. Wattenhofer, "Achieving High Utilization with Software-driven WAN," ACM *SIGCOMM Conference* (Aug. 2013), pp.15–26.

[Huang 2002] C. Haung, V. Sharma, K. Owens, V. Makam, "Building Reliable MPLS Networks Using a Path Protection Mechanism," *IEEE Communications Magazine*, Vol. 40, No. 3 (Mar. 2002), pp. 156–162.

[Huang 2005] Y. Huang, R. Guerin, "Does Over-Provisioning Become More or Less Efficient as Networks Grow Larger?," *Proc. IEEE Int. Conf. Network Protocols (ICNP)* (Boston MA, Nov. 2005).

[Huang 2008] C. Huang, J. Li, A. Wang, K. W. Ross, "Understanding Hybrid CDN-P2P: Why Limelight Needs Its Own Red Swoosh," *Proc. 2008 NOSSDAV*, Braunschweig, Germany.

[Huitema 1998] C. Huitema, *IPv6: The New Internet Protocol*, 2nd Ed., Prentice Hall, Englewood Cliffs, NJ, 1998.

[Huston 1999a] G. Huston, "Interconnection, Peering, and Settlements—Part I," *The Internet Protocol Journal*, Vol. 2, No. 1 (Mar. 1999).

[Huston 2004] G. Huston, "NAT Anatomy: A Look Inside Network Address Translators," *The Internet Protocol Journal*, Vol. 7, No. 3 (Sept. 2004).

[Huston 2008a] G. Huston, "Confronting IPv4 Address Exhaustion," http://www.potaroo.net/ispcol/2008-10/v4depletion.html

[Huston 2008b] G. Huston, G. Michaelson, "IPv6 Deployment: Just where are we?" http://www.potaroo.net/ispcol/2008-04/ipv6.html

[Huston 2011a] G. Huston, "A Rough Guide to Address Exhaustion," *The Internet Protocol Journal*, Vol. 14, No. 1 (Mar. 2011).

[Huston 2011b] G. Huston, "Transitioning Protocols," *The Internet Protocol Journal*, Vol. 14, No. 1 (Mar. 2011).

[IAB 2016] Internet Architecture Board homepage, http://www.iab.org/

[IANA Protocol Numbers 2016] Internet Assigned Numbers Authority, Protocol Numbers, http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml

[IBM 1997] IBM Corp., *IBM Inside APPN - The Essential Guide to the Next-Generation SNA*, SG24-3669-03, June 1997.

[ICANN 2016] The Internet Corporation for Assigned Names and Numbers homepage, http://www.icann.org

[IEEE 802 2016] IEEE 802 LAN/MAN Standards Committee homepage, http://www.ieee802.org/

[IEEE 802.11 1999] IEEE 802.11, "1999 Edition (ISO/IEC 8802-11: 1999) IEEE Standards for Information Technology—Telecommunications and Information Exchange Between Systems—Local and Metropolitan Area Network—Specific Requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification," http://standards.ieee.org/getieee802/download/802.11-1999.pdf

[IEEE 802.11ac 2013] IEEE, "802.11ac-2013—IEEE Standard for Information technology—Telecommunications and Information Exchange Between Systems—Local and Metropolitan Area Networks—Specific Requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications—Amendment 4: Enhancements for Very High Throughput for Operation in Bands Below 6 GHz."

[IEEE 802.11n 2012] IEEE, "IEEE P802.11—Task Group N—Meeting Update: Status of 802.11n," http://grouper.ieee.org/groups/802/11/Reports/tgn_update.htm

[IEEE 802.15 2012] IEEE 802.15 Working Group for WPAN homepage, http://grouper.ieee.org/groups/802/15/.

[IEEE 802.15.4 2012] IEEE 802.15 WPAN Task Group 4, http://www.ieee802.org/15/pub/TG4.html

[IEEE 802.16d 2004] IEEE, "IEEE Standard for Local and Metropolitan Area Networks, Part 16: Air Interface for Fixed Broadband Wireless Access Systems," http://standards.ieee.org/getieee802/download/802.16-2004.pdf

[IEEE 802.16e 2005] IEEE, "IEEE Standard for Local and Metropolitan Area Networks, Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems, Amendment 2: Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands and Corrigendum 1," http://standards.ieee.org/getieee802/download/802.16e-2005.pdf

[IEEE 802.1q 2005] IEEE, "IEEE Standard for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks," http://standards.ieee.org/getieee802/download/802.1Q-2005.pdf

[IEEE 802.1X] IEEE Std 802.1X-2001 Port-Based Network Access Control, http://standards.ieee.org/reading/ieee/std_public/description/lanman/802.1x-2001_desc.html

[IEEE 802.3 2012] IEEE, "IEEE 802.3 CSMA/CD (Ethernet)," http://grouper.ieee.org/groups/802/3/

[IEEE 802.5 2012] IEEE, IEEE 802.5 homepage, http://www.ieee802.org/5/www8025org/

[IETF 2016] Internet Engineering Task Force homepage, http://www.ietf.org

[Ihm 2011] S. Ihm, V. S. Pai, "Towards Understanding Modern Web Traffic," *Proc. 2011 ACM Internet Measurement Conference* (Berlin).

[IMAP 2012] The IMAP Connection, http://www.imap.org/

[Intel 2016] Intel Corp., "Intel 710 Ethernet Adapter," http://www.intel.com/content/www/us/en/ethernet-products/converged-network-adapters/ethernet-xl710.html

[Internet2 Multicast 2012] Internet2 Multicast Working Group homepage, http://www.internet2.edu/multicast/

[ISC 2016] Internet Systems Consortium homepage, http://www.isc.org

[ISI 1979] Information Sciences Institute, "DoD Standard Internet Protocol," Internet Engineering Note 123 (Dec. 1979), http://www.isi.edu/in-notes/ien/ien123.txt

[ISO 2016] International Organization for Standardization homepage, International Organization for Standardization, http://www.iso.org/

[ISO X.680 2002] International Organization for Standardization, "X.680: ITU-T Recommendation X.680 (2002) Information Technology—Abstract Syntax Notation One (ASN.1): Specification of Basic Notation," http://www.itu.int/ITU-T/studygroups/com17/languages/X.680-0207.pdf

[ITU 1999] Asymmetric Digital Subscriber Line (ADSL) Transceivers. ITU-T G.992.1, 1999.

[ITU 2003] Asymmetric Digital Subscriber Line (ADSL) Transceivers—Extended Bandwidth ADSL2 (ADSL2Plus). ITU-T G.992.5, 2003.

[ITU 2005a] International Telecommunication Union, "ITU-T X.509, The Directory: Public-key and attribute certificate frameworks" (Aug. 2005).

[ITU 2006] ITU, "G.993.1: Very High Speed Digital Subscriber Line Transceivers (VDSL)," https://www.itu.int/rec/T-REC-G.993.1-200406-I/en, 2006.

[ITU 2015] "Measuring the Information Society Report," 2015, http://www.itu.int/en/ITU-D/Statistics/Pages/publications/mis2015.aspx

[ITU 2012] The ITU homepage, http://www.itu.int/

[ITU-T Q.2931 1995] International Telecommunication Union, "Recommendation Q.2931 (02/95)—Broadband Integrated Services Digital Network (B-ISDN)—Digital Subscriber Signalling System No. 2 (DSS 2)—User-Network Interface (UNI)—Layer 3 Specification for Basic Call/Connection Control."

[IXP List 2016] List of IXPs, Wikipedia, https://en.wikipedia.org/wiki/List_of_Internet_exchange_points

[Iyengar 2015] J. Iyengar, I. Swett, "QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2," Internet Draft draft-tsvwg-quic-protocol-00, June 2015.

[Iyer 2008] S. Iyer, R. R. Kompella, N. McKeown, "Designing Packet Buffers for Router Line Cards," *IEEE Transactions on Networking*, Vol. 16, No. 3 (June 2008), pp. 705–717.

[Jacobson 1988] V. Jacobson, "Congestion Avoidance and Control," *Proc. 1988 ACM SIGCOMM* (Stanford, CA, Aug. 1988), pp. 314–329.

[Jain 1986] R. Jain, "A Timeout-Based Congestion Control Scheme for Window Flow-Controlled Networks," *IEEE Journal on Selected Areas in Communications SAC-4*, 7 (Oct. 1986).

[Jain 1989] R. Jain, "A Delay-Based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks," *ACM SIGCOMM Computer Communications Review*, Vol. 19, No. 5 (1989), pp. 56–71.

[Jain 1994] R. Jain, *FDDI Handbook: High-Speed Networking Using Fiber and Other Media*, Addison-Wesley, Reading, MA, 1994.

[Jain 1996] R. Jain. S. Kalyanaraman, S. Fahmy, R. Goyal, S. Kim, "Tutorial Paper on ABR Source Behavior," *ATM Forum*/96-1270, Oct. 1996. http://www.cse.wustl.edu/~jain/atmf/ftp/atm96-1270.pdf

[Jain 2013] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S.Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, A, Vahdat, "B4: Experience with a Globally Deployed Software Defined Wan," *ACM SIGCOMM 2013*, pp. 3–14.

[Jaiswal 2003] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, D. Towsley, "Measurement and Classification of Out-of-Sequence Packets in a Tier-1 IP backbone," *Proc. 2003 IEEE INFOCOM*.

[Ji 2003] P. Ji, Z. Ge, J. Kurose, D. Towsley, "A Comparison of Hard-State and Soft-State Signaling Protocols," *Proc. 2003 ACM SIGCOMM* (Karlsruhe, Germany, Aug. 2003).

[Jimenez 1997] D. Jimenez, "Outside Hackers Infiltrate MIT Network, Compromise Security," *The Tech*, Vol. 117, No 49 (Oct. 1997), p. 1, http://www-tech.mit.edu/V117/N49/hackers.49n.html

[Jin 2004] C. Jin, D. X. We, S. Low, "FAST TCP: Motivation, Architecture, Algorithms, Performance," *Proc. 2004 IEEE INFOCOM* (Hong Kong, Mar. 2004).

[Juniper Contrail 2016] Juniper Networks, "Contrail," http://www.juniper.net/us/en/products-services/sdn/contrail/

[Juniper MX2020 2015] Juniper Networks, "MX2020 and MX2010 3D Universal Edge Routers," www.juniper.net/us/en/local/pdf/.../1000417-en.pdf

[Kaaranen 2001] H. Kaaranen, S. Naghian, L. Laitinen, A. Ahtiainen, V. Niemi, *Networks: Architecture, Mobility and Services*, New York: John Wiley & Sons, 2001.

[Kahn 1967] D. Kahn, *The Codebreakers: The Story of Secret Writing*, The Macmillan Company, 1967.

[Kahn 1978] R. E. Kahn, S. Gronemeyer, J. Burchfiel, R. Kunzelman, "Advances in Packet Radio Technology," *Proc. 1978 IEEE INFOCOM*, 66, 11 (Nov. 1978).

[Kamerman 1997] A. Kamerman, L. Monteban, "WaveLAN-II: A High–Performance Wireless LAN for the Unlicensed Band," *Bell Labs Technical Journal* (Summer 1997), pp. 118–133.

[Kar 2000] K. Kar, M. Kodialam, T. V. Lakshman, "Minimum Interference Routing of Bandwidth Guaranteed Tunnels with MPLS Traffic Engineering Applications," *IEEE J. Selected Areas in Communications* (Dec. 2000).

[Karn 1987] P. Karn, C. Partridge, "Improving Round-Trip Time Estimates in Reliable Transport Protocols," *Proc. 1987 ACM SIGCOMM*.

[Karol 1987] M. Karol, M. Hluchyj, A. Morgan, "Input Versus Output Queuing on a Space-Division Packet Switch," *IEEE Transactions on Communications*, Vol. 35, No. 12 (Dec.1987), pp. 1347–1356.

[Kaufman 1995] C. Kaufman, R. Perlman, M. Speciner, *Network Security, Private Communication in a Public World*, Prentice Hall, Englewood Cliffs, NJ, 1995.

[Kelly 1998] F. P. Kelly, A. Maulloo, D. Tan, "Rate Control for Communication Networks: Shadow Prices, Proportional Fairness and Stability," *J. Operations Res. Soc.*, Vol. 49, No. 3 (Mar. 1998), pp. 237–252.

[Kelly 2003] T. Kelly, "Scalable TCP: Improving Performance in High Speed Wide Area Networks," *ACM SIGCOMM Computer Communications Review*, Volume 33, No. 2 (Apr. 2003), pp 83–91.

[Kilkki 1999] K. Kilkki, *Differentiated Services for the Internet*, Macmillan Technical Publishing, Indianapolis, IN, 1999.

[Kim 2005] H. Kim, S. Rixner, V. Pai, "Network Interface Data Caching," *IEEE Transactions on Computers*, Vol. 54, No. 11 (Nov. 2005), pp. 1394–1408.

[Kim 2008] C. Kim, M. Caesar, J. Rexford, "Floodless in SEATTLE: A Scalable Ethernet Architecture for Large Enterprises," *Proc. 2008 ACM SIGCOMM* (Seattle, WA, Aug. 2008).

[Kleinrock 1961] L. Kleinrock, "Information Flow in Large Communication Networks," RLE Quarterly Progress Report, July 1961.

[Kleinrock 1964] L. Kleinrock, *1964 Communication Nets: Stochastic Message Flow and Delay*, McGraw-Hill, New York, NY, 1964.

[Kleinrock 1975] L. Kleinrock, *Queuing Systems, Vol. 1*, John Wiley, New York, 1975.

[Kleinrock 1975b] L. Kleinrock, F. A. Tobagi, "Packet Switching in Radio Channels: Part I—Carrier Sense Multiple-Access Modes and Their Throughput-Delay Characteristics," *IEEE Transactions on Communications*, Vol. 23, No. 12 (Dec. 1975), pp. 1400–1416.

[Kleinrock 1976] L. Kleinrock, *Queuing Systems, Vol. 2*, John Wiley, New York, 1976.

[Kleinrock 2004] L. Kleinrock, "The Birth of the Internet," http://www.lk.cs.ucla.edu/LK/Inet/birth.html

[Kohler 2006] E. Kohler, M. Handley, S. Floyd, "DDCP: Designing DCCP: Congestion Control Without Reliability," *Proc. 2006 ACM SIGCOMM* (Pisa, Italy, Sept. 2006).

[Kolding 2003] T. Kolding, K. Pedersen, J. Wigard, F. Frederiksen, P. Mogensen, "High Speed Downlink Packet Access: WCDMA Evolution," *IEEE Vehicular Technology Society News* (Feb. 2003), pp. 4–10.

[Koponen 2010] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, S. Shenker, "Onix: A Distributed Control Platform for Large-Scale Production Networks," *9th USENIX conference on Operating systems design and implementation (OSDI'10)*, pp. 1–6.

[Koponen 2011] T. Koponen, S. Shenker, H. Balakrishnan, N. Feamster, I. Ganichev, A. Ghodsi, P. B. Godfrey, N. McKeown, G. Parulkar, B. Raghavan, J. Rexford, S. Arianfar, D. Kuptsov, "Architecting for Innovation," *ACM Computer Communications Review*, 2011.

[Korhonen 2003] J. Korhonen, *Introduction to 3G Mobile Communications*, 2nd ed., Artech House, 2003.

[Koziol 2003] J. Koziol, *Intrusion Detection with Snort*, Sams Publishing, 2003.

[Kreutz 2015] D. Kreutz, F.M.V. Ramos, P. Esteves Verissimo, C. Rothenberg, S. Azodolmolky, S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, Vol. 103, No. 1 (Jan. 2015), pp. 14-76. This paper is also being updated at https://github.com/SDN-Survey/latex/wiki

[Krishnamurthy 2001] B. Krishnamurthy, J. Rexford, *Web Protocols and Practice: HTTP/ 1.1, Networking Protocols, and Traffic Measurement*, Addison-Wesley, Boston, MA, 2001.

[Kulkarni 2005] S. Kulkarni, C. Rosenberg, "Opportunistic Scheduling: Generalizations to Include Multiple Constraints, Multiple Interfaces, and Short Term Fairness," *Wireless Networks*, 11 (2005), 557–569.

[Kumar 2006] R. Kumar, K.W. Ross, "Optimal Peer-Assisted File Distribution: Single and Multi-Class Problems," *IEEE Workshop on Hot Topics in Web Systems and Technologies* (Boston, MA, 2006).

[Labovitz 1997] C. Labovitz, G. R. Malan, F. Jahanian, "Internet Routing Instability," *Proc. 1997 ACM SIGCOMM* (Cannes, France, Sept. 1997), pp. 115–126.

[Labovitz 2010] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, F. Jahanian, "Internet Inter-Domain Traffic," *Proc. 2010 ACM SIGCOMM*.

[Labrador 1999] M. Labrador, S. Banerjee, "Packet Dropping Policies for ATM and IP Networks," *IEEE Communications Surveys*, Vol. 2, No. 3 (Third Quarter 1999), pp. 2–14.

[Lacage 2004] M. Lacage, M.H. Manshaei, T. Turletti, "IEEE 802.11 Rate Adaptation: A Practical Approach," *ACM Int. Symposium on Modeling, Analysis, and Simulation of Wireless and Mobile Systems (MSWiM)* (Venice, Italy, Oct. 2004).

[Lakhina 2004] A. Lakhina, M. Crovella, C. Diot, "Diagnosing Network-Wide Traffic Anomalies," *Proc. 2004 ACM SIGCOMM*.

[Lakhina 2005] A. Lakhina, M. Crovella, C. Diot, "Mining Anomalies Using Traffic Feature Distributions," *Proc. 2005 ACM SIGCOMM*.

[Lakshman 1997] T. V. Lakshman, U. Madhow, "The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss," *IEEE/ACM Transactions on Networking*, Vol. 5, No. 3 (1997), pp. 336–350.

[Lakshman 2004] T. V. Lakshman, T. Nandagopal, R. Ramjee, K. Sabnani, T. Woo, "The SoftRouter Architecture," *Proc. 3nd ACM Workshop on Hot Topics in Networks (Hotnets-III)*, Nov. 2004.

[Lam 1980] S. Lam, "A Carrier Sense Multiple Access Protocol for Local Networks," *Computer Networks*, Vol. 4 (1980), pp. 21–32.

[Lamport 1989] L. Lamport, "The Part-Time Parliament," Technical Report 49, Systems Research Center, Digital Equipment Corp., Palo Alto, Sept. 1989.

[Lampson 1983] Lampson, Butler W. "Hints for computer system design," *ACM SIGOPS Operating Systems Review*, Vol. 17, No. 5, 1983.

[Lampson 1996] B. Lampson, "How to Build a Highly Available System Using Consensus," *Proc. 10th International Workshop on Distributed Algorithms* (WDAG '96), Özalp Babaoglu and Keith Marzullo (Eds.), Springer-Verlag, pp. 1–17.

[Lawton 2001] G. Lawton, "Is IPv6 Finally Gaining Ground?" *IEEE Computer Magazine* (Aug. 2001), pp. 11–15.

[LeBlond 2011] S. Le Blond, C. Zhang, A. Legout, K. Ross, W. Dabbous. 2011, "I know where you are and what you are sharing: exploiting P2P communications to invade users' privacy." *2011 ACM Internet Measurement Conference*, ACM, New York, NY, USA, pp. 45–60.

[Leighton 2009] T. Leighton, "Improving Performance on the Internet," *Communications of the ACM*, Vol. 52, No. 2 (Feb. 2009), pp. 44–51.

[Leiner 1998] B. Leiner, V. Cerf, D. Clark, R. Kahn, L. Kleinrock, D. Lynch, J. Postel, L. Roberts, S. Woolf, "A Brief History of the Internet," http://www.isoc.org/internet/history/brief.html

[Leung 2006] K. Leung, V. O.K. Li, "TCP in Wireless Networks: Issues, Approaches, and Challenges," *IEEE Commun. Surveys and Tutorials*, Vol. 8, No. 4 (2006), pp. 64–79.

[Levin 2012] D. Levin, A. Wundsam, B. Heller, N. Handigol, A. Feldmann, "Logically Centralized?: State Distribution Trade-offs in Software Defined Networks," *Proc. First Workshop on Hot Topics in Software Defined Networks* (Aug. 2012), pp. 1–6.

[Li 2004] L. Li, D. Alderson, W. Willinger, J. Doyle, "A First-Principles Approach to Understanding the Internet's Router-Level Topology," *Proc. 2004 ACM SIGCOMM* (Portland, OR, Aug. 2004).

[Li 2007] J. Li, M. Guidero, Z. Wu, E. Purpus, T. Ehrenkranz, "BGP Routing Dynamics Revisited." *ACM Computer Communication Review* (Apr. 2007).

[Li 2015] S.Q. Li, "Building Softcom Ecosystem Foundation," Open Networking Summit, 2015.

[Lin 2001] Y. Lin, I. Chlamtac, *Wireless and Mobile Network Architectures*, John Wiley and Sons, New York, NY, 2001.

[Liogkas 2006] N. Liogkas, R. Nelson, E. Kohler, L. Zhang, "Exploiting BitTorrent for Fun (but Not Profit)," *6th International Workshop on Peer-to-Peer Systems (IPTPS 2006).*

[Liu 2003] J. Liu, I. Matta, M. Crovella, "End-to-End Inference of Loss Nature in a Hybrid Wired/Wireless Environment," *Proc. WiOpt'03: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks.*

[Locher 2006] T. Locher, P. Moor, S. Schmid, R. Wattenhofer, "Free Riding in BitTorrent is Cheap," *Proc. ACM HotNets 2006* (Irvine CA, Nov. 2006).

[Lui 2004] J. Lui, V. Misra, D. Rubenstein, "On the Robustness of Soft State Protocols," *Proc. IEEE Int. Conference on Network Protocols (ICNP '04)*, pp. 50–60.

[Mahdavi 1997] J. Mahdavi, S. Floyd, "TCP-Friendly Unicast Rate-Based Flow Control," unpublished note (Jan. 1997).

[MaxMind 2016] http://www.maxmind.com/app/ip-location

[Maymounkov 2002] P. Maymounkov, D. Mazières. "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric." *Proceedings of the 1st International Workshop on Peerto-Peer Systems (IPTPS '02)* (Mar. 2002), pp. 53–65.

[McKeown 1997a] N. McKeown, M. Izzard, A. Mekkittikul, W. Ellersick, M. Horowitz, "The Tiny Tera: A Packet Switch Core," *IEEE Micro Magazine* (Jan.–Feb. 1997).

[McKeown 1997b] N. McKeown, "A Fast Switched Backplane for a Gigabit Switched Router," *Business Communications Review*, Vol. 27, No. 12. http://tinytera.stanford.edu/~nickm/papers/cisco_fasts_wp.pdf

[McKeown 2008] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner. 2008. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.* 38, 2 (Mar. 2008), pp. 69–74.

[McQuillan 1980] J. McQuillan, I. Richer, E. Rosen, "The New Routing Algorithm for the Arpanet," *IEEE Transactions on Communications*, Vol. 28, No. 5 (May 1980), pp. 711–719.

[Metcalfe 1976] R. M. Metcalfe, D. R. Boggs. "Ethernet: Distributed Packet Switching for Local Computer Networks," *Communications of the Association for Computing Machinery*, Vol. 19, No. 7 (July 1976), pp. 395–404.

[Meyers 2004] A. Myers, T. Ng, H. Zhang, "Rethinking the Service Model: Scaling Ethernet to a Million Nodes," *ACM Hotnets Conference*, 2004.

[MFA Forum 2016] IP/MPLS Forum homepage, http://www.ipmplsforum.org/

[Mockapetris 1988] P. V. Mockapetris, K. J. Dunlap, "Development of the Domain Name System," *Proc. 1988 ACM SIGCOMM* (Stanford, CA, Aug. 1988).

[Mockapetris 2005] P. Mockapetris, Sigcomm Award Lecture, video available at http://www.postel.org/sigcomm

[Molinero-Fernandez 2002] P. Molinaro-Fernandez, N. McKeown, H. Zhang, "Is IP Going to Take Over the World (of Communications)?" *Proc. 2002 ACM Hotnets*.

[Molle 1987] M. L. Molle, K. Sohraby, A. N. Venetsanopoulos, "Space-Time Models of Asynchronous CSMA Protocols for Local Area Networks," *IEEE Journal on Selected Areas in Communications*, Vol. 5, No. 6 (1987), pp. 956–968.

[Moore 2001] D. Moore, G. Voelker, S. Savage, "Inferring Internet Denial of Service Activity," *Proc. 2001 USENIX Security Symposium* (Washington, DC, Aug. 2001).

[Motorola 2007] Motorola, "Long Term Evolution (LTE): A Technical Overview," http://www.motorola.com/staticfiles/Business/Solutions/Industry%20Solutions/Service%20Providers/Wireless%20Operators/LTE/_Document/Static%20Files/6834_MotDoc_New.pdf

[Mouly 1992] M. Mouly, M. Pautet, *The GSM System for Mobile Communications*, Cell and Sys, Palaiseau, France, 1992.

[Moy 1998] J. Moy, *OSPF: Anatomy of An Internet Routing Protocol*, Addison-Wesley, Reading, MA, 1998.

[Mukherjee 1997] B. Mukherjee, *Optical Communication Networks*, McGraw-Hill, 1997.

[Mukherjee 2006] B. Mukherjee, *Optical WDM Networks*, Springer, 2006.

[Mysore 2009] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, A. Vahdat, "PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric," *Proc. 2009 ACM SIGCOMM*.

[Nahum 2002] E. Nahum, T. Barzilai, D. Kandlur, "Performance Issues in WWW Servers," *IEEE/ACM Transactions on Networking*, Vol 10, No. 1 (Feb. 2002).

[Netflix Open Connect 2016] Netflix Open Connect CDN, 2016, https://openconnect.netflix.com/

[Netflix Video 1] Designing Netflix's Content Delivery System, D. Fulllager, 2014, https://www.youtube.com/watch?v=LkLLpYdDINA

[Netflix Video 2] Scaling the Netflix Global CDN, D. Temkin, 2015, https://www.youtube.com/watch?v=tbqcsHg-Q_o

[Neumann 1997] R. Neumann, "Internet Routing Black Hole," *The Risks Digest: Forum on Risks to the Public in Computers and Related Systems*, Vol. 19, No. 12 (May 1997). http://catless.ncl.ac.uk/Risks/19.12.html#subj1.1

[Neville-Neil 2009] G. Neville-Neil, "Whither Sockets?" *Communications of the ACM*, Vol. 52, No. 6 (June 2009), pp. 51–55.

[Nicholson 2006] A Nicholson, Y. Chawathe, M. Chen, B. Noble, D. Wetherall, "Improved Access Point Selection," *Proc. 2006 ACM Mobisys Conference* (Uppsala Sweden, 2006).

[Nielsen 1997] H. F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. W. Lie, C. Lilley, "Network Performance Effects of HTTP/1.1, CSS1, and PNG," *W3C Document*, 1997 (also appears in *Proc. 1997 ACM SIGCOM* (Cannes, France, Sept 1997), pp. 155–166.

[NIST 2001] National Institute of Standards and Technology, "Advanced Encryption Standard (AES)," Federal Information Processing Standards 197, Nov. 2001, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[NIST IPv6 2015] US National Institute of Standards and Technology, "Estimating IPv6 & DNSSEC Deployment SnapShots," http://fedv6-deployment.antd.nist.gov/snap-all.html

[Nmap 2012] Nmap homepage, http://www.insecure.com/nmap

[Nonnenmacher 1998] J. Nonnenmacher, E. Biersak, D. Towsley, "Parity-Based Loss Recovery for Reliable Multicast Transmission," *IEEE/ACM Transactions on Networking*, Vol. 6, No. 4 (Aug. 1998), pp. 349–361.

[Nygren 2010] Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun, "The Akamai Network: A Platform for High-performance Internet Applications," *SIGOPS Oper. Syst. Rev.* 44, 3 (Aug. 2010), 2–19.

[ONF 2016] Open Networking Foundation, Technical Library, https://www.opennetworking.org/sdn-resources/technical-library

[ONOS 2016] Open Network Operating System (ONOS), "Architecture Guide," https://wiki.onosproject.org/display/ONOS/Architecture+Guide, 2016.

[OpenFlow 2009] Open Network Foundation, "OpenFlow Switch Specification 1.0.0, TS-001," https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf

[OpenDaylight Lithium 2016] OpenDaylight, "Lithium," https://www.opendaylight.org/lithium

[OSI 2012] International Organization for Standardization homepage, http://www.iso.org/iso/en/ISOOnline.frontpage

[Osterweil 2012] E. Osterweil, D. McPherson, S. DiBenedetto, C. Papadopoulos, D. Massey, "Behavior of DNS Top Talkers," *Passive and Active Measurement Conference*, 2012.

[Padhye 2000] J. Padhye, V. Firoiu, D. Towsley, J. Kurose, "Modeling TCP Reno Performance: A Simple Model and Its Empirical Validation," *IEEE/ACM Transactions on Networking*, Vol. 8 No. 2 (Apr. 2000), pp. 133–145.

[Padhye 2001] J. Padhye, S. Floyd, "On Inferring TCP Behavior," *Proc. 2001 ACM SIGCOMM* (San Diego, CA, Aug. 2001).

[Palat 2009] S. Palat, P. Godin, "The LTE Network Architecture: A Comprehensive Tutorial," in *LTE—The UMTS Long Term Evolution: From Theory to Practice.* Also available as a standalone Alcatel white paper.

[Panda 2013] A. Panda, C. Scott, A. Ghodsi, T. Koponen, S. Shenker, "CAP for Networks," *Proc. ACM HotSDN '13*, pp. 91–96.

[Parekh 1993] A. Parekh, R. Gallagher, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case," *IEEE/ACM Transactions on Networking*, Vol. 1, No. 3 (June 1993), pp. 344–357.

[Partridge 1992] C. Partridge, S. Pink, "An Implementation of the Revised Internet Stream Protocol (ST-2)," *Journal of Internetworking: Research and Experience*, Vol. 3, No. 1 (Mar. 1992).

[Partridge 1998] C. Partridge, et al. "A Fifty Gigabit per second IP Router," *IEEE/ACM Transactions on Networking*, Vol. 6, No. 3 (Jun. 1998), pp. 237–248.

[Pathak 2010] A. Pathak, Y. A. Wang, C. Huang, A. Greenberg, Y. C. Hu, J. Li, K. W. Ross, "Measuring and Evaluating TCP Splitting for Cloud Services," *Passive and Active Measurement (PAM) Conference* (Zurich, 2010).

[Perkins 1994] A. Perkins, "Networking with Bob Metcalfe," *The Red Herring Magazine* (Nov. 1994).

[Perkins 1998] C. Perkins, O. Hodson, V. Hardman, "A Survey of Packet Loss Recovery Techniques for Streaming Audio," *IEEE Network Magazine* (Sept./Oct. 1998), pp. 40–47.

[Perkins 1998b] C. Perkins, *Mobile IP: Design Principles and Practice*, Addison-Wesley, Reading, MA, 1998.

[Perkins 2000] C. Perkins, *Ad Hoc Networking*, Addison-Wesley, Reading, MA, 2000.

[Perlman 1999] R. Perlman, *Interconnections: Bridges, Routers, Switches, and Internetworking Protocols*, 2nd ed., Addison-Wesley Professional Computing Series, Reading, MA, 1999.

[PGPI 2016] The International PGP homepage, http://www.pgpi.org

[Phifer 2000] L. Phifer, "The Trouble with NAT," *The Internet Protocol Journal*, Vol. 3, No. 4 (Dec. 2000), http://www.cisco.com/warp/public/759/ipj_3-4/ipj_3-4_nat.html

[Piatek 2007] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, A. Venkataramani, "Do Incentives Build Robustness in Bittorrent?," *Proc. NSDI* (2007).

[Piatek 2008] M. Piatek, T. Isdal, A. Krishnamurthy, T. Anderson, "One Hop Reputations for Peer-to-peer File Sharing Workloads," *Proc. NSDI* (2008).

[Pickholtz 1982] R. Pickholtz, D. Schilling, L. Milstein, "Theory of Spread Spectrum Communication—a Tutorial," *IEEE Transactions on Communications*, Vol. 30, No. 5 (May 1982), pp. 855–884.

[PingPlotter 2016] PingPlotter homepage, http://www.pingplotter.com

[Piscatello 1993] D. Piscatello, A. Lyman Chapin, *Open Systems Networking*, Addison-Wesley, Reading, MA, 1993.

[Pomeranz 2010] H. Pomeranz, "Practical, Visual, Three-Dimensional Pedagogy for Internet Protocol Packet Header Control Fields," https://righteousit.wordpress.com/2010/06/27/practical-visual-three-dimensional-pedagogy-for-internet-protocol-packet-header-control-fields/, June 2010.

[Potaroo 2016] "Growth of the BGP Table–1994 to Present," http://bgp.potaroo.net/

[PPLive 2012] PPLive homepage, http://www.pplive.com

[Qazi 2013] Z. Qazi, C. Tu, L. Chiang, R. Miao, V. Sekar, M. Yu, "SIMPLE-fying Middlebox Policy Enforcement Using SDN," ACM *SIGCOMM Conference* (Aug. 2013), pp. 27–38.

[Quagga 2012] Quagga, "Quagga Routing Suite," http://www.quagga.net/

[Quittner 1998] J. Quittner, M. Slatalla, *Speeding the Net: The Inside Story of Netscape and How It Challenged Microsoft*, Atlantic Monthly Press, 1998.

[Quova 2016] www.quova.com

[Ramakrishnan 1990] K. K. Ramakrishnan, R. Jain, "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks," *ACM Transactions on Computer Systems*, Vol. 8, No. 2 (May 1990), pp. 158–181.

[Raman 1999] S. Raman, S. McCanne, "A Model, Analysis, and Protocol Framework for Soft State-based Communication," *Proc. 1999 ACM SIGCOMM* (Boston, MA, Aug. 1999).

[Raman 2007] B. Raman, K. Chebrolu, "Experiences in Using WiFi for Rural Internet in India," *IEEE Communications Magazine*, Special Issue on New Directions in Networking Technologies in Emerging Economies (Jan. 2007).

[Ramaswami 2010] R. Ramaswami, K. Sivarajan, G. Sasaki, *Optical Networks: A Practical Perspective*, Morgan Kaufman Publishers, 2010.

[Ramjee 1994] R. Ramjee, J. Kurose, D. Towsley, H. Schulzrinne, "Adaptive Playout Mechanisms for Packetized Audio Applications in Wide-Area Networks," *Proc. 1994 IEEE INFOCOM.*

[Rao 2011] A. S. Rao, Y. S. Lim, C. Barakat, A. Legout, D. Towsley, W. Dabbous, "Network Characteristics of Video Streaming Traffic," *Proc. 2011 ACM CoNEXT* (Tokyo).

[Ren 2006] S. Ren, L. Guo, X. Zhang, "ASAP: An AS-Aware Peer-Relay Protocol for High Quality VoIP," *Proc. 2006 IEEE ICDCS* (Lisboa, Portugal, July 2006).

[Rescorla 2001] E. Rescorla, *SSL and TLS: Designing and Building Secure Systems*, Addison-Wesley, Boston, 2001.

[RFC 001] S. Crocker, "Host Software," RFC 001 (the *very first* RFC!).

[RFC 768] J. Postel, "User Datagram Protocol," RFC 768, Aug. 1980.

[RFC 791] J. Postel, "Internet Protocol: DARPA Internet Program Protocol Specification," RFC 791, Sept. 1981.

[RFC 792] J. Postel, "Internet Control Message Protocol," RFC 792, Sept. 1981.

[RFC 793] J. Postel, "Transmission Control Protocol," RFC 793, Sept. 1981.

[RFC 801] J. Postel, "NCP/TCP Transition Plan," RFC 801, Nov. 1981.

[RFC 826] D. C. Plummer, "An Ethernet Address Resolution Protocol—or—Converting Network Protocol Addresses to 48-bit Ethernet Address for Transmission on Ethernet Hardware," RFC 826, Nov. 1982.

[RFC 829] V. Cerf, "Packet Satellite Technology Reference Sources," RFC 829, Nov. 1982.

[RFC 854] J. Postel, J. Reynolds, "TELNET Protocol Specification," RFC 854, May 1993.

[RFC 950] J. Mogul, J. Postel, "Internet Standard Subnetting Procedure," RFC 950, Aug. 1985.

[RFC 959] J. Postel and J. Reynolds, "File Transfer Protocol (FTP)," RFC 959, Oct. 1985.

[RFC 1034] P. V. Mockapetris, "Domain Names—Concepts and Facilities," RFC 1034, Nov. 1987.

[RFC 1035] P. Mockapetris, "Domain Names—Implementation and Specification," RFC 1035, Nov. 1987.

[RFC 1058] C. L. Hendrick, "Routing Information Protocol," RFC 1058, June 1988.

[RFC 1071] R. Braden, D. Borman, and C. Partridge, "Computing the Internet Checksum," RFC 1071, Sept. 1988.

[RFC 1122] R. Braden, "Requirements for Internet Hosts—Communication Layers," RFC 1122, Oct. 1989.

[RFC 1123] R. Braden, ed., "Requirements for Internet Hosts—Application and Support," RFC-1123, Oct. 1989.

[RFC 1142] D. Oran, "OSI IS-IS Intra-Domain Routing Protocol," RFC 1142, Feb. 1990.

[RFC 1190] C. Topolcic, "Experimental Internet Stream Protocol: Version 2 (ST-II)," RFC 1190, Oct. 1990.

[RFC 1256] S. Deering, "ICMP Router Discovery Messages," RFC 1256, Sept. 1991.

[RFC 1320] R. Rivest, "The MD4 Message-Digest Algorithm," RFC 1320, Apr. 1992.

[RFC 1321] R. Rivest, "The MD5 Message-Digest Algorithm," RFC 1321, Apr. 1992.

[RFC 1323] V. Jacobson, S. Braden, D. Borman, "TCP Extensions for High Performance," RFC 1323, May 1992.

[RFC 1422] S. Kent, "Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management," RFC 1422.

[RFC 1546] C. Partridge, T. Mendez, W. Milliken, "Host Anycasting Service," RFC 1546, 1993.

[RFC 1584] J. Moy, "Multicast Extensions to OSPF," RFC 1584, Mar. 1994.

[RFC 1633] R. Braden, D. Clark, S. Shenker, "Integrated Services in the Internet Architecture: an Overview," RFC 1633, June 1994.

[RFC 1636] R. Braden, D. Clark, S. Crocker, C. Huitema, "Report of IAB Workshop on Security in the Internet Architecture," RFC 1636, Nov. 1994.

[RFC 1700] J. Reynolds, J. Postel, "Assigned Numbers," RFC 1700, Oct. 1994.

[RFC 1752] S. Bradner, A. Mankin, "The Recommendations for the IP Next Generation Protocol," RFC 1752, Jan. 1995.

[RFC 1918] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, E. Lear, "Address Allocation for Private Internets," RFC 1918, Feb. 1996.

[RFC 1930] J. Hawkinson, T. Bates, "Guidelines for Creation, Selection, and Registration of an Autonomous System (AS)," RFC 1930, Mar. 1996.

[RFC 1939] J. Myers, M. Rose, "Post Office Protocol—Version 3," RFC 1939, May 1996.

[RFC 1945] T. Berners-Lee, R. Fielding, H. Frystyk, "Hypertext Transfer Protocol—HTTP/1.0," RFC 1945, May 1996.

[RFC 2003] C. Perkins, "IP Encapsulation Within IP," RFC 2003, Oct. 1996.

[RFC 2004] C. Perkins, "Minimal Encapsulation Within IP," RFC 2004, Oct. 1996.

[RFC 2018] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, "TCP Selective Acknowledgment Options," RFC 2018, Oct. 1996.

[RFC 2131] R. Droms, "Dynamic Host Configuration Protocol," RFC 2131, Mar. 1997.

[RFC 2136] P. Vixie, S. Thomson, Y. Rekhter, J. Bound, "Dynamic Updates in the Domain Name System," RFC 2136, Apr. 1997.

[RFC 2205] R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, S. Jamin, "Resource ReSerVation Protocol (RSVP)—Version 1 Functional Specification," RFC 2205, Sept. 1997.

[RFC 2210] J. Wroclawski, "The Use of RSVP with IETF Integrated Services," RFC 2210, Sept. 1997.

[RFC 2211] J. Wroclawski, "Specification of the Controlled-Load Network Element Service," RFC 2211, Sept. 1997.

[RFC 2215] S. Shenker, J. Wroclawski, "General Characterization Parameters for Integrated Service Network Elements," RFC 2215, Sept. 1997.

[RFC 2326] H. Schulzrinne, A. Rao, R. Lanphier, "Real Time Streaming Protocol (RTSP)," RFC 2326, Apr. 1998.

[RFC 2328] J. Moy, "OSPF Version 2," RFC 2328, Apr. 1998.

[RFC 2420] H. Kummert, "The PPP Triple-DES Encryption Protocol (3DESE)," RFC 2420, Sept. 1998.

[RFC 2453] G. Malkin, "RIP Version 2," RFC 2453, Nov. 1998.

[RFC 2460] S. Deering, R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," RFC 2460, Dec. 1998.

[RFC 2475] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, "An Architecture for Differentiated Services," RFC 2475, Dec. 1998.

[RFC 2578] K. McCloghrie, D. Perkins, J. Schoenwaelder, "Structure of Management Information Version 2 (SMIv2)," RFC 2578, Apr. 1999.

[RFC 2579] K. McCloghrie, D. Perkins, J. Schoenwaelder, "Textual Conventions for SMIv2," RFC 2579, Apr. 1999.

[RFC 2580] K. McCloghrie, D. Perkins, J. Schoenwaelder, "Conformance Statements for SMIv2," RFC 2580, Apr. 1999.

[RFC 2597] J. Heinanen, F. Baker, W. Weiss, J. Wroclawski, "Assured Forwarding PHB Group," RFC 2597, June 1999.

[RFC 2616] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, R. Fielding, "Hypertext Transfer Protocol—HTTP/1.1," RFC 2616, June 1999.

[RFC 2663] P. Srisuresh, M. Holdrege, "IP Network Address Translator (NAT) Terminology and Considerations," RFC 2663.

[RFC 2702] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, J. McManus, "Requirements for Traffic Engineering Over MPLS," RFC 2702, Sept. 1999.

[RFC 2827] P. Ferguson, D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which Employ IP Source Address Spoofing," RFC 2827, May 2000.

[RFC 2865] C. Rigney, S. Willens, A. Rubens, W. Simpson, "Remote Authentication Dial In User Service (RADIUS)," RFC 2865, June 2000.

[RFC 3007] B. Wellington, "Secure Domain Name System (DNS) Dynamic Update," RFC 3007, Nov. 2000.

[RFC 3022] P. Srisuresh, K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)," RFC 3022, Jan. 2001.

[RFC 3022] P. Srisuresh, K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)," RFC 3022, Jan. 2001.

[RFC 3031] E. Rosen, A. Viswanathan, R. Callon, "Multiprotocol Label Switching Architecture," RFC 3031, Jan. 2001.

[RFC 3032] E. Rosen, D. Tappan, G. Fedorkow, Y. Rekhter, D. Farinacci, T. Li, A. Conta, "MPLS Label Stack Encoding," RFC 3032, Jan. 2001.

[RFC 3168] K. Ramakrishnan, S. Floyd, D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," RFC 3168, Sept. 2001.

[RFC 3209] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, G. Swallow, "RSVP-TE: Extensions to RSVP for LSP Tunnels," RFC 3209, Dec. 2001.

[RFC 3221] G. Huston, "Commentary on Inter-Domain Routing in the Internet," RFC 3221, Dec. 2001.

[RFC 3232] J. Reynolds, "Assigned Numbers: RFC 1700 Is Replaced by an Online Database," RFC 3232, Jan. 2002.

[RFC 3234] B. Carpenter, S. Brim, "Middleboxes: Taxonomy and Issues," RFC 3234, Feb. 2002.

[RFC 3246] B. Davie, A. Charny, J.C.R. Bennet, K. Benson, J.Y. Le Boudec, W. Courtney, S. Davari, V. Firoiu, D. Stiliadis, "An Expedited Forwarding PHB (Per-Hop Behavior)," RFC 3246, Mar. 2002.

[RFC 3260] D. Grossman, "New Terminology and Clarifications for Diffserv," RFC 3260, Apr. 2002.

[RFC 3261] J. Rosenberg, H. Schulzrinne, G. Carmarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, "SIP: Session Initiation Protocol," RFC 3261, July 2002.

[RFC 3272] J. Boyle, V. Gill, A. Hannan, D. Cooper, D. Awduche, B. Christian, W. S. Lai, "Overview and Principles of Internet Traffic Engineering," RFC 3272, May 2002.

[RFC 3286] L. Ong, J. Yoakum, "An Introduction to the Stream Control Transmission Protocol (SCTP)," RFC 3286, May 2002.

[RFC 3346] J. Boyle, V. Gill, A. Hannan, D. Cooper, D. Awduche, B. Christian, W. S. Lai, "Applicability Statement for Traffic Engineering with MPLS," RFC 3346, Aug. 2002.

[RFC 3390] M. Allman, S. Floyd, C. Partridge, "Increasing TCP's Initial Window," RFC 3390, Oct. 2002.

[RFC 3410] J. Case, R. Mundy, D. Partain, "Introduction and Applicability Statements for Internet Standard Management Framework," RFC 3410, Dec. 2002.

[RFC 3414] U. Blumenthal and B. Wijnen, "User-based Security Model (USM) for Version 3 of the Simple Network Management Protocol (SNMPv3)," RFC 3414, Dec. 2002.

[RFC 3416] R. Presuhn, J. Case, K. McCloghrie, M. Rose, S. Waldbusser, "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)," Dec. 2002.

[RFC 3439] R. Bush, D. Meyer, "Some Internet Architectural Guidelines and Philosophy," RFC 3439, Dec. 2003.

[RFC 3447] J. Jonsson, B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1," RFC 3447, Feb. 2003.

[RFC 3468] L. Andersson, G. Swallow, "The Multiprotocol Label Switching (MPLS) Working Group Decision on MPLS Signaling Protocols," RFC 3468, Feb. 2003.

[RFC 3469] V. Sharma, Ed., F. Hellstrand, Ed, "Framework for Multi-Protocol Label Switching (MPLS)-based Recovery," RFC 3469, Feb. 2003. ftp://ftp.rfc-editor.org/in-notes/rfc3469.txt

[RFC 3501] M. Crispin, "Internet Message Access Protocol—Version 4rev1," RFC 3501, Mar. 2003.

[RFC 3550] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," RFC 3550, July 2003.

[RFC 3588] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, J. Arkko, "Diameter Base Protocol," RFC 3588, Sept. 2003.

[RFC 3649] S. Floyd, "HighSpeed TCP for Large Congestion Windows," RFC 3649, Dec. 2003.

[RFC 3746] L. Yang, R. Dantu, T. Anderson, R. Gopal, "Forwarding and Control Element Separation (ForCES) Framework," Internet, RFC 3746, Apr. 2004.

[RFC 3748] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, H. Levkowetz, Ed., "Extensible Authentication Protocol (EAP)," RFC 3748, June 2004.

[RFC 3782] S. Floyd, T. Henderson, A. Gurtov, "The NewReno Modification to TCP's Fast Recovery Algorithm," RFC 3782, Apr. 2004.

[RFC 4213] E. Nordmark, R. Gilligan, "Basic Transition Mechanisms for IPv6 Hosts and Routers," RFC 4213, Oct. 2005.

[RFC 4271] Y. Rekhter, T. Li, S. Hares, Ed., "A Border Gateway Protocol 4 (BGP-4)," RFC 4271, Jan. 2006.

[RFC 4272] S. Murphy, "BGP Security Vulnerabilities Analysis," RFC 4274, Jan. 2006.

[RFC 4291] R. Hinden, S. Deering, "IP Version 6 Addressing Architecture," RFC 4291, Feb. 2006.

[RFC 4340] E. Kohler, M. Handley, S. Floyd, "Datagram Congestion Control Protocol (DCCP)," RFC 4340, Mar. 2006.

[RFC 4443] A. Conta, S. Deering, M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification," RFC 4443, Mar. 2006.

[RFC 4346] T. Dierks, E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1," RFC 4346, Apr. 2006.

[RFC 4514] K. Zeilenga, Ed., "Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names," RFC 4514, June 2006.

[RFC 4601] B. Fenner, M. Handley, H. Holbrook, I. Kouvelas, "Protocol Independent Multicast—Sparse Mode (PIM-SM): Protocol Specification (Revised)," RFC 4601, Aug. 2006.

[RFC 4632] V. Fuller, T. Li, "Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan," RFC 4632, Aug. 2006.

[RFC 4960] R. Stewart, ed., "Stream Control Transmission Protocol," RFC 4960, Sept. 2007.

[RFC 4987] W. Eddy, "TCP SYN Flooding Attacks and Common Mitigations," RFC 4987, Aug. 2007.

[RFC 5000] RFC editor, "Internet Official Protocol Standards," RFC 5000, May 2008.

[RFC 5109] A. Li (ed.), "RTP Payload Format for Generic Forward Error Correction," RFC 5109, Dec. 2007.

[RFC 5216] D. Simon, B. Aboba, R. Hurst, "The EAP-TLS Authentication Protocol," RFC 5216, Mar. 2008.

[RFC 5218] D. Thaler, B. Aboba, "What Makes for a Successful Protocol?," RFC 5218, July 2008.

[RFC 5321] J. Klensin, "Simple Mail Transfer Protocol," RFC 5321, Oct. 2008.

[RFC 5322] P. Resnick, Ed., "Internet Message Format," RFC 5322, Oct. 2008.

[RFC 5348] S. Floyd, M. Handley, J. Padhye, J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification," RFC 5348, Sept. 2008.

[RFC 5389] J. Rosenberg, R. Mahy, P. Matthews, D. Wing, "Session Traversal Utilities for NAT (STUN)," RFC 5389, Oct. 2008.

[RFC 5411] J Rosenberg, "A Hitchhiker's Guide to the Session Initiation Protocol (SIP)," RFC 5411, Feb. 2009.

[RFC 5681] M. Allman, V. Paxson, E. Blanton, "TCP Congestion Control," RFC 5681, Sept. 2009.

[RFC 5944] C. Perkins, Ed., "IP Mobility Support for IPv4, Revised," RFC 5944, Nov. 2010.

[RFC 6265] A Barth, "HTTP State Management Mechanism," RFC 6265, Apr. 2011.

[RFC 6298] V. Paxson, M. Allman, J. Chu, M. Sargent, "Computing TCP's Retransmission Timer," RFC 6298, June 2011.

[RFC 7020] R. Housley, J. Curran, G. Huston, D. Conrad, "The Internet Numbers Registry System," RFC 7020, Aug. 2013.

[RFC 7094] D. McPherson, D. Oran, D. Thaler, E. Osterweil, "Architectural Considerations of IP Anycast," RFC 7094, Jan. 2014.

[RFC 7323] D. Borman, R. Braden, V. Jacobson, R. Scheffenegger (ed.), "TCP Extensions for High Performance," RFC 7323, Sept. 2014.

[RFC 7540] M. Belshe, R. Peon, M. Thomson (Eds), "Hypertext Transfer Protocol Version 2 (HTTP/2)," RFC 7540, May 2015.

[Richter 2015] P. Richter, M. Allman, R. Bush, V. Paxson, "A Primer on IPv4 Scarcity," *ACM SIGCOMM Computer Communication Review*, Vol. 45, No. 2 (Apr. 2015), pp. 21–32.

[Roberts 1967] L. Roberts, T. Merril, "Toward a Cooperative Network of Time-Shared Computers," *AFIPS Fall Conference* (Oct. 1966).

[Rodriguez 2010] R. Rodrigues, P. Druschel, "Peer-to-Peer Systems," *Communications of the ACM*, Vol. 53, No. 10 (Oct. 2010), pp. 72–82.

[Rohde 2008] Rohde, Schwarz, "UMTS Long Term Evolution (LTE) Technology Introduction," Application Note 1MA111.

[Rom 1990] R. Rom, M. Sidi, *Multiple Access Protocols: Performance and Analysis*, Springer-Verlag, New York, 1990.

[Root Servers 2016] Root Servers home page, http://www.root-servers.org/

[RSA 1978] R. Rivest, A. Shamir, L. Adelman, "A Method for Obtaining Digital Signatures and Public-key Cryptosystems," *Communications of the ACM*, Vol. 21, No. 2 (Feb. 1978), pp. 120–126.

[RSA Fast 2012] RSA Laboratories, "How Fast Is RSA?" http://www.rsa.com/rsalabs/node.asp?id=2215

[RSA Key 2012] RSA Laboratories, "How Large a Key Should Be Used in the RSA Crypto System?" http://www.rsa.com/rsalabs/node.asp?id=2218

[Rubenstein 1998] D. Rubenstein, J. Kurose, D. Towsley, "Real-Time Reliable Multicast Using Proactive Forward Error Correction," *Proceedings of NOSSDAV '98* (Cambridge, UK, July 1998).

[Ruiz-Sanchez 2001] M. Ruiz-Sánchez, E. Biersack, W. Dabbous, "Survey and Taxonomy of IP Address Lookup Algorithms," *IEEE Network Magazine*, Vol. 15, No. 2 (Mar./Apr. 2001), pp. 8–23.

[Saltzer 1984] J. Saltzer, D. Reed, D. Clark, "End-to-End Arguments in System Design," *ACM Transactions on Computer Systems (TOCS)*, Vol. 2, No. 4 (Nov. 1984).

[Sandvine 2015] "Global Internet Phenomena Report, Spring 2011," http://www.sandvine.com/news/global broadband trends.asp, 2011.

[Sardar 2006] B. Sardar, D. Saha, "A Survey of TCP Enhancements for Last-Hop Wireless Networks," *IEEE Commun. Surveys and Tutorials*, Vol. 8, No. 3 (2006), pp. 20–34.

[Saroiu 2002] S. Saroiu, P. K. Gummadi, S. D. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," *Proc. of Multimedia Computing and Networking (MMCN)* (2002).

[Sauter 2014] M. Sauter, *From GSM to LTE-Advanced*, John Wiley and Sons, 2014.

[Savage 2015] D. Savage, J. Ng, S. Moore, D. Slice, P. Paluch, R. White, "Enhanced Interior Gateway Routing Protocol," Internet Draft, draft-savage-eigrp-04.txt, Aug. 2015.

[Saydam 1996] T. Saydam, T. Magedanz, "From Networks and Network Management into Service and Service Management," *Journal of Networks and System Management*, Vol. 4, No. 4 (Dec. 1996), pp. 345–348.

[Schiller 2003] J. Schiller, *Mobile Communications* 2nd edition, Addison Wesley, 2003.

[Schneier 1995] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, John Wiley and Sons, 1995.

[Schulzrinne-RTP 2012] Henning Schulzrinne's RTP site, http://www.cs.columbia.edu/~hgs/rtp

[Schulzrinne-SIP 2016] Henning Schulzrinne's SIP site, http://www.cs.columbia.edu/~hgs/sip

[Schwartz 1977] M. Schwartz, *Computer-Communication Network Design and Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1997.

[Schwartz 1980] M. Schwartz, *Information, Transmission, Modulation, and Noise,* McGraw Hill, New York, NY 1980.

[Schwartz 1982] M. Schwartz, "Performance Analysis of the SNA Virtual Route Pacing Control," *IEEE Transactions on Communications*, Vol. 30, No. 1 (Jan. 1982), pp. 172–184.

[Scourias 2012] J. Scourias, "Overview of the Global System for Mobile Communications: GSM." http://www.privateline.com/PCS/GSM0.html

[SDNHub 2016] SDNHub, "App Development Tutorials," http://sdnhub.org/tutorials/

[Segaller 1998] S. Segaller, *Nerds 2.0.1, A Brief History of the Internet*, TV Books, New York, 1998.

[Sekar 2011] V. Sekar, S. Ratnasamy, M. Reiter, N. Egi, G. Shi, " The Middlebox Manifesto: Enabling Innovation in Middlebox Deployment," *Proc. 10th ACM Workshop on Hot Topics in Networks (HotNets)*, Article 21, 6 pages.

[Serpanos 2011] D. Serpanos, T. Wolf, *Architecture of Network Systems*, Morgan Kaufmann Publishers, 2011.

[Shacham 1990] N. Shacham, P. McKenney, "Packet Recovery in High-Speed Networks Using Coding and Buffer Management," *Proc. 1990 IEEE INFOCOM* (San Francisco, CA, Apr. 1990), pp. 124–131.

[Shaikh 2001] A. Shaikh, R. Tewari, M. Agrawal, "On the Effectiveness of DNS-based Server Selection," *Proc. 2001 IEEE INFOCOM*.

[Singh 1999] S. Singh, *The Code Book: The Evolution of Secrecy from Mary, Queen of Scotsto Quantum Cryptography*, Doubleday Press, 1999.

[Singh 2015] A. Singh, J. Ong,. Agarwal, G. Anderson, A. Armistead, R. Banno, S. Boving, G. Desai, B. Felderman, P. Germano, A. Kanagala, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart, A. Vahdat, "Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network," Sigcomm, 2015.

[SIP Software 2016] H. Schulzrinne Software Package site, http://www.cs.columbia.edu/IRT/software

[Skoudis 2004] E. Skoudis, L. Zeltser, *Malware: Fighting Malicious Code*, Prentice Hall, 2004.

[Skoudis 2006] E. Skoudis, T. Liston, *Counter Hack Reloaded: A Step-by-Step Guide to Computer Attacks and Effective Defenses (2nd Edition)*, Prentice Hall, 2006.

[Smith 2009] J. Smith, "Fighting Physics: A Tough Battle," *Communications of the ACM*, Vol. 52, No. 7 (July 2009), pp. 60–65.

[Snort 2012] Sourcefire Inc., Snort homepage, http://http://www.snort.org/

[Solensky 1996] F. Solensky, "IPv4 Address Lifetime Expectations," in *IPng: Internet Protocol Next Generation* (S. Bradner, A. Mankin, ed.), Addison-Wesley, Reading, MA, 1996.

[Spragins 1991] J. D. Spragins, *Telecommunications Protocols and Design*, Addison-Wesley, Reading, MA, 1991.

[Srikant 2004] R. Srikant, *The Mathematics of Internet Congestion Control*, Birkhauser, 2004

[Steinder 2002] M. Steinder, A. Sethi, "Increasing Robustness of Fault Localization Through Analysis of Lost, Spurious, and Positive Symptoms," *Proc. 2002 IEEE INFOCOM.*

[Stevens 1990] W. R. Stevens, *Unix Network Programming*, Prentice-Hall, Englewood Cliffs, NJ.

[Stevens 1994] W. R. Stevens, *TCP/IP Illustrated, Vol. 1: The Protocols*, Addison-Wesley, Reading, MA, 1994.

[Stevens 1997] W.R. Stevens, *Unix Network Programming, Volume 1: Networking APIs-Sockets and XTI*, 2nd edition, Prentice-Hall, Englewood Cliffs, NJ, 1997.

[Stewart 1999] J. Stewart, *BGP4: Interdomain Routing in the Internet*, Addison-Wesley, 1999.

[Stone 1998] J. Stone, M. Greenwald, C. Partridge, J. Hughes, "Performance of Checksums and CRC's Over Real Data," *IEEE/ACM Transactions on Networking*, Vol. 6, No. 5 (Oct. 1998), pp. 529–543.

[Stone 2000] J. Stone, C. Partridge, "When Reality and the Checksum Disagree," *Proc. 2000 ACM SIGCOMM* (Stockholm, Sweden, Aug. 2000).

[Strayer 1992] W. T. Strayer, B. Dempsey, A. Weaver, *XTP: The Xpress Transfer Protocol*, Addison-Wesley, Reading, MA, 1992.

[Stubblefield 2002] A. Stubblefield, J. Ioannidis, A. Rubin, "Using the Fluhrer, Mantin, and Shamir Attack to Break WEP," *Proceedings of 2002 Network and Distributed Systems Security Symposium* (2002), pp. 17–22.

[Subramanian 2000] M. Subramanian, *Network Management: Principles and Practice*, Addison-Wesley, Reading, MA, 2000.

[Subramanian 2002] L. Subramanian, S. Agarwal, J. Rexford, R. Katz, "Characterizing the Internet Hierarchy from Multiple Vantage Points," *Proc. 2002 IEEE INFOCOM*.

[Sundaresan 2006] K.Sundaresan, K. Papagiannaki, "The Need for Cross-layer Information in Access Point Selection," *Proc. 2006 ACM Internet Measurement Conference* (Rio De Janeiro, Oct. 2006).

[Suh 2006] K. Suh, D. R. Figueiredo, J. Kurose and D. Towsley, "Characterizing and Detecting Relayed Traffic: A Case Study Using Skype," *Proc. 2006 IEEE INFOCOM* (Barcelona, Spain, Apr. 2006).

[Sunshine 1978] C. Sunshine, Y. Dalal, "Connection Management in Transport Protocols," *Computer Networks*, North-Holland, Amsterdam, 1978.

[Tariq 2008] M. Tariq, A. Zeitoun, V. Valancius, N. Feamster, M. Ammar, "Answering What-If Deployment and Configuration Questions with WISE," *Proc. 2008 ACM SIGCOMM* (Aug. 2008).

[TechnOnLine 2012] TechOnLine, "Protected Wireless Networks," online webcast tutorial, http://www.techonline.com/community/tech_topic/internet/21752

[Teixeira 2006] R. Teixeira, J. Rexford, "Managing Routing Disruptions in Internet Service Provider Networks," *IEEE Communications Magazine* (Mar. 2006).

[Think 2012] Technical History of Network Protocols, "Cyclades," http://www.cs.utexas.edu/users/chris/think/Cyclades/index.shtml

[Tian 2012] Y. Tian, R. Dey, Y. Liu, K. W. Ross, "China's Internet: Topology Mapping and Geolocating," *IEEE INFOCOM Mini-Conference 2012* (Orlando, FL, 2012).

[TLD list 2016] TLD list maintained by Wikipedia, https://en.wikipedia.org/wiki/List_of_Internet_top-level_domains

[Tobagi 1990] F. Tobagi, "Fast Packet Switch Architectures for Broadband Integrated Networks," *Proc. 1990 IEEE INFOCOM*, Vol. 78, No. 1 (Jan. 1990), pp. 133–167.

[TOR 2016] Tor: Anonymity Online, http://www.torproject.org

[Torres 2011] R. Torres, A. Finamore, J. R. Kim, M. M. Munafo, S. Rao, "Dissecting Video Server Selection Strategies in the YouTube CDN," *Proc. 2011 Int. Conf. on Distributed Computing Systems*.

[Tourrilhes 2014] J. Tourrilhes, P. Sharma, S. Banerjee, J. Petit, "SDN and Openflow Evolution: A Standards Perspective," *IEEE Computer Magazine*, Nov. 2014, pp. 22–29.

[Turner 1988] J. S. Turner, "Design of a Broadcast packet switching network," *IEEE Transactions on Communications*, Vol. 36, No. 6 (June 1988), pp. 734–743.

[Turner 2012] B. Turner, "2G, 3G, 4G Wireless Tutorial," http://blogs.nmscommunications.com/communications/2008/10/2g-3g-4g-wireless-tutorial.html

[UPnP Forum 2016] UPnP Forum homepage, http://www.upnp.org/

[van der Berg 2008] R. van der Berg, "How the 'Net Works: An Introduction to Peering and Transit," http://arstechnica.com/guides/other/peering-and-transit.ars

[van der Merwe 1998] J. van der Merwe, S. Rooney, I. Leslie, S. Crosby, "The Tempest: A Practical Framework for Network Programmability," *IEEE Network*, Vol. 12, No. 3 (May 1998), pp. 20–28.

[Varghese 1997] G. Varghese, A. Lauck, "Hashed and Hierarchical Timing Wheels: Efficient Data Structures for Implementing a Timer Facility," *IEEE/ACM Transactions on Networking*, Vol. 5, No. 6 (Dec. 1997), pp. 824–834.

[Vasudevan 2012] S. Vasudevan, C. Diot, J. Kurose, D. Towsley, "Facilitating Access Point Selection in IEEE 802.11 Wireless Networks," *Proc. 2005 ACM Internet Measurement Conference*, (San Francisco CA, Oct. 2005).

[Villamizar 1994] C. Villamizar, C. Song. "High Performance TCP in ANSNET," *ACM SIGCOMM Computer Communications Review*, Vol. 24, No. 5 (1994), pp. 45–60.

[Viterbi 1995] A. Viterbi, *CDMA: Principles of Spread Spectrum Communication*, Addison-Wesley, Reading, MA, 1995.

[Vixie 2009] P. Vixie, "What DNS Is Not," *Communications of the ACM*, Vol. 52, No. 12 (Dec. 2009), pp. 43–47.

[Wakeman 1992] I. Wakeman, J. Crowcroft, Z. Wang, D. Sirovica, "Layering Considered Harmful," *IEEE Network* (Jan. 1992), pp. 20–24.

[Waldrop 2007] M. Waldrop, "Data Center in a Box," *Scientific American* (July 2007).

[Wang 2004] B. Wang, J. Kurose, P. Shenoy, D. Towsley, "Multimedia Streaming via TCP: An Analytic Performance Study," *Proc. 2004 ACM Multimedia Conference* (New York, NY, Oct. 2004).

[Wang 2008] B. Wang, J. Kurose, P. Shenoy, D. Towsley, "Multimedia Streaming via TCP: An Analytic Performance Study," *ACM Transactions on Multimedia Computing Communications and Applications (TOMCCAP)*, Vol. 4, No. 2 (Apr. 2008), p. 16. 1–22.

[Wang 2010] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. S. E. Ng, M. Kozuch, M. Ryan, "c-Through: Part-time Optics in Data Centers," *Proc. 2010 ACM SIGCOMM*.

[Wei 2006] W. Wei, C. Zhang, H. Zang, J. Kurose, D. Towsley, "Inference and Evaluation of Split-Connection Approaches in Cellular Data Networks," *Proc. Active and Passive Measurement Workshop* (Adelaide, Australia, Mar. 2006).

[Wei 2007] D. X. Wei, C. Jin, S. H. Low, S. Hegde, "FAST TCP: Motivation, Architecture, Algorithms, Performance," *IEEE/ACM Transactions on Networking* (2007).

[Weiser 1991] M. Weiser, "The Computer for the Twenty-First Century," *Scientific American* (Sept. 1991): 94–10. http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html

[White 2011] A. White, K. Snow, A. Matthews, F. Monrose, "Hookt on fon-iks: Phonotactic Reconstruction of Encrypted VoIP Conversations," *IEEE Symposium on Security and Privacy*, Oakland, CA, 2011.

[Wigle.net 2016] Wireless Geographic Logging Engine, http://www.wigle.net

[Wiki Satellite 2016] Satellite Internet access, https://en.wikipedia.org/wiki/Satellite_Internet_access

[Wireshark 2016] Wireshark homepage, http://www.wireshark.org

[Wischik 2005] D. Wischik, N. McKeown, "Part I: Buffer Sizes for Core Routers," *ACM SIGCOMM Computer Communications Review*, Vol. 35, No. 3 (July 2005).

[Woo 1994] T. Woo, R. Bindignavle, S. Su, S. Lam, "SNP: an interface for secure network programming," *Proc. 1994 Summer USENIX* (Boston, MA, June 1994), pp. 45–58.

[Wright 2015] J. Wright, J. *Wireless Security Secrets & Solutions*, 3e, "Hacking Exposed Wireless," McGraw-Hill Education, 2015.

[Wu 2005] J. Wu, Z. M. Mao, J. Rexford, J. Wang, "Finding a Needle in a Haystack: Pinpointing Significant BGP Routing Changes in an IP Network," *Proc. USENIX NSDI* (2005).

[Xanadu 2012] Xanadu Project homepage, http://www.xanadu.com/

[Xiao 2000] X. Xiao, A. Hannan, B. Bailey, L. Ni, "Traffic Engineering with MPLS in the Internet," *IEEE Network* (Mar./Apr. 2000).

[Xu 2004] L. Xu, K Harfoush, I. Rhee, "Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks," *IEEE INFOCOM 2004*, pp. 2514–2524.

[Yavatkar 1994] R. Yavatkar, N. Bhagwat, "Improving End-to-End Performance of TCP over Mobile Internetworks," *Proc. Mobile 94 Workshop on Mobile Computing Systems and Applications* (Dec. 1994).

[YouTube 2009] YouTube 2009, Google container data center tour, 2009.

[YouTube 2016] YouTube Statistics, 2016, https://www.youtube.com/yt/press/statistics.html

[Yu 2004] Yu, Fang, H. Katz, Tirunellai V. Lakshman. "Gigabit Rate Packet Pattern-Matching Using TCAM," *Proc. 2004 Int. Conf. Network Protocols*, pp. 174–183.

[Yu 2011] M. Yu, J. Rexford, X. Sun, S. Rao, N. Feamster, "A Survey of VLAN Usage in Campus Networks," *IEEE Communications Magazine*, July 2011.

[Zegura 1997] E. Zegura, K. Calvert, M. Donahoo, "A Quantitative Comparison of Graph-based Models for Internet Topology," *IEEE/ACM Transactions on Networking*, Vol. 5, No. 6, (Dec. 1997). See also http://www.cc.gatech.edu/projects/gtim for a software package that generates networks with a transit-stub structure.

[Zhang 1993] L. Zhang, S. Deering, D. Estrin, S. Shenker, D. Zappala, "RSVP: A New Resource Reservation Protocol," *IEEE Network Magazine*, Vol. 7, No. 9 (Sept. 1993), pp. 8–18.

[Zhang 2007] L. Zhang, "A Retrospective View of NAT," *The IETF Journal*, Vol. 3, Issue 2 (Oct. 2007).

[Zhang 2015] G. Zhang, W. Liu, X. Hei, W. Cheng, "Unreeling Xunlei Kankan: Understanding Hybrid CDN-P2P Video-on-Demand Streaming," *IEEE Transactions on Multimedia*, Vol. 17, No. 2, Feb. 2015.

[Zhang X 2102] X. Zhang, Y. Xu, Y. Liu, Z. Guo, Y. Wang, "Profiling Skype Video Calls: Rate Control and Video Quality," *IEEE INFOCOM* (Mar. 2012).

[Zink 2009] M. Zink, K. Suh, Y. Gu, J. Kurose, "Characteristics of YouTube Network Traffic at a Campus Network—Measurements, Models, and Implications," *Computer Networks*, Vol. 53, No. 4, pp. 501–514, 2009.

Index

A

AAC. *See* Advanced Audio Coding
ABR. *See* ATM Available Bit Rate
Abramson, Norman, 62, 460, 478
access control lists, 654, 656
access networks, 12–18, 404
  cable, 14–15, 64, 463, 465–467
  dial-up access, 16, 491
  DSL, 13–14, 64
  enterprise, 16–17
  Ethernet, 16–17
  FTTH, 15–16, 64
  HFC, 14
  LTE, 18, 552, 553, 557–560
  radio access networks, 556–560
  satellite, 16, 439
  3G, 18
  WiFi, 16–17
access points (APs), 522, 523
  in infrastructure LANs, 534
  MAC addresses, 533, 535
  mobility between, 560–561
  power management and, 548
  scanning for, 536
ACK (positive acknowledgments), 210–214
  corrupted, 212
  DHCP, 344–345, 502
  duplicate, 214, 249
  in 802.11 RTC/CTS system, 541–542
  TCP generation recommendation, 250
ACK bit, 236
  TCP, 653–654
ack clocking, 303
ACK frames, 541–542
acknowledged segments, 271
acknowledgement frames, 539
acknowledgements
  cumulative, 224, 238
  link-layer, 537, 538, 539
  negative, 210–214, 241, 448
  piggybacked, 241
  positive, 210–214, 249, 344–345, 541–542
  TCP, 237–239, 252
acknowledgment number, 237–239
  piggybacked, 241
  Telnet and, 239–241
acknowledgment number field, 236
ACK received events, 245, 246
active optical networks (AONs), 15
active queue management (AQM), 324
active scanning, 536
adapters, 443, 444
  ARP query and, 472
  CSMA/CD operation and, 462–463
  datagram transmission and, 471, 473–474
  802.11, 533, 537
  error detection in, 448
  Ethernet frames and, 476–478
  jabbering, 484
  LAN-on-motherboard configuration, 443
  layer independence and, 470
  MAC addresses, 468–470
  on separate cards, 443

adaptive backoff, 516
adaptive congestion control, 203
adaptive HTTP streaming, 681
adaptive playout delay, 692–694
additive-increase, multiplicative-decrease (AIMD), 277
  fairness of, 279–282
address aggregation, 339
addresses. *See also* IP addresses; MAC addresses
  anycast, 349
  care-of, 564, 565–567, 573
  foreign, 564
  IP broadcast, 341, 343
  LAN, 468
  MAC broadcast, 470, 472
  mobile node, 561
  obtaining with DHCP, 342–345
  permanent, 564
  physical, 468
  realm with private, 345
  SIP, 705–706
addressing, 89–90
  classful, 338–339
  IPv4, 334–345
  link-layer, 468–474
  mobility and, 562–564
  mobility management and, 562–564
  processes, 89–90
  SIP, 705–706
address lease time, 344
Address Resolution Protocol (ARP), 468, 470–473, 503
  data center network design and, 497
Address Supporting Organization of ICANN, 341
ad hoc networks, 534
  mobile, 524–525, 562
  vehicular, 525
Adleman, Leonard, 605
administrative autonomy, 392
Advanced Audio Coding (AAC), 678, 700
Advanced Research Projects Agency (ARPA), 60, 371, 516
AES (Advanced Encryption Standard), 602, 636
agent advertisement, 571, 574
agent discovery, 570, 571–573
agent solicitation, 572
aging time, 483
AH protocol. *See* Authentication Header protocol
AIMD. *See* additive-increase, multiplicative-decrease
Akamai, 150, 155
aliasing
  host, 128, 136
  mail server, 128
ALOHANet, 60, 62, 460
ALOHA protocols, 455, 478
  backoff in, 516
  pure, 458
  slotted, 456–458, 516
alternating-bit protocol, 217, 218
Alto computers, 478
Amazon, 64, 500, 675
  cloud services, 154
  Netflix and, 154–156
  video streaming, 147, 679
anchor foreign agent, 569
anchor MSC, 579
Andreessen, Marc, 63, 184–185
Andreessen Horowitz, 184
Android devices, 18
anomaly-based systems, 661
anonymity, 658

anycast address, 349
AONs. *See* active optical networks
Apache Web server, 158, 199
API. *See* Application Programming Interface
application architecture, 86
application delay, 43
application gateways, 652, 656–657, 659
application layer, 50
application-layer message, 54
application-layer protocols, 95
  defining, 96–97
  DNS, 50
  FTP, 50
  HTTP, 50, 96–97
  Skype, 96
  SMTP, 50, 97
application-level transport reliability, 203–204
Application Programming Interface (API), 89
application protocols, well-known, 195
applications. *See also* multimedia applications; network applications
  bandwidth-sensitive, 92
  delay-sensitive, 680
  distributed, 5
  elastic, 92
  loss-tolerant, 91, 680
  network-service, 416
  peer-to-peer, 140–147
  real-time conversational, 700–708
  SDN control, 410–412
  transport services available to, 90–93
APs. *See* access points
AQM. *See* active queue management
A records, 135
ARP. *See* Address Resolution Protocol
ARPA. *See* Advanced Research Projects Agency
ARPAnet, 234
  ALOHAnet connection to, 460
  Cerf on, 371
  development of, 60–63
  Lam on, 516
  Metcalfe and, 478
  routing algorithms, 379, 386
ARPAnet Satellite System, 516
ARP packet, 472
ARP query, 503
ARP reply, 503
ARP table, 472
ARQ (Automatic Repeat reQuest) protocols, 210, 448
ASN. *See* autonomous system number
AS numbers. *See* autonomous system number
AS-PATH, 399, 401
ASs. *See* autonomous systems
associate, 535
association
  in 802.11, 534–837
  security, 640–641, 645
assured forwarding PHB, 722
Atheros AR5006, 443
ATM, 517
  congestion control, 268
  delay and bandwidth guarantees, 312
  Ethernet and, 475
  IP device interconnection with, 492
  MPLS headers for, 492
  Q2931b and, 725
  QoS information in, 725
  SDN and, 415
ATM Available Bit Rate (ABR), 261
  congestion control, 268

AT&T, 5, 375, 436, 736
audio
AAC, 678, 700
jitter removal for, 691–694
MP3, 678, 700
properties of, 677–679
quantization, 678
RTP payloads, 703
Skype quality adaptation for, 697
streaming, 679–680
authentication
802.11i, 649–650
end-point, 58–59, 595
MD5, 394
message, code for, 613–617, 634–635, 642
in OSPF, 394
receiver, 627
sender, 627, 628
simple, 394
wireless LANs, 536–537
Authentication Header protocol (AH protocol), 640
authentication key, 613
authentication protocol, 621–625
authoritative DNS servers, 132, 504
autonomous system number (ASN), 392
autonomous systems (ASs), 392
in BGP route advertisement, 396–398
hierarchy within, 394–395
iBGP connections within, 397
routing between, 391–395, 405, 416
availability zones, 500
average throughput, 44
Azure, 65

B

B4, 375, 413, 416
backbone providers, 404
backoff
adaptive, 516
binary exponential, 463
random, 539
in slotted ALOHA, 516
back pressure, 686
bandwidth, 28–29
application sensitivity to, 92
ATM guarantees, 312
best-effort service and, 312
bottleneck, 111
congestion control and, 271
fairness and, 279–282
guaranteed minimal, 311–312
host-to-host, 498, 499
HTTP streaming and, 148–149
link-layer switching properties and, 484
memory, 319
P2P applications and, 140, 142
QoS guarantees, 710, 725
Skype usage of, 699
TCP and high, 278–279
throughput and, 91–92
traffic class isolation and, 715–716
UDP streaming and, 681, 683
video early termination wasting, 688
video prefetching and, 684
video properties and, 676–677
video streaming quality adaptation to, 680
Web caching and, 111–112
bandwidth flooding, 56, 57, 139
bandwidth probing, 271, 277
bandwidth provisioning, 711
bandwidth-sensitive applications, 92
Baran, Paul, 60

base HTML file, 98
base station (BS), 522, 533
  handoff and, 577–579
base station controller (BSC), 554
base station subsystem (BSS), 554
base transceiver station (BTS), 554
basic service set (BSS), 533, 544
  mobility across, 546–547
BBN, 61
beacon frames, 535, 536, 548
beacon signals, 578
Bellman-Ford equation, 384–385
Bellovin, Steven, 673–674
BER. *See* bit error rate
Berkeley Internet Name Domain (BIND), 127
Berners-Lee, Tim, 63
best-effort delivery services, 192
best-effort services, 312
  dimensioning, 709, 711–712
  limitations of, 688–689
  multiple classes of service for, 712–719
BGP. *See* Border Gateway Protocol
BGP attributes, 398–399
BGP connection, 397
bidirectional data transfer, 208
binary exponential backoff, 463
BIND. *See* Berkeley Internet Name Domain
bind(), 195
bit error rate (BER), 526–528
  rate adaptation and, 547
bit errors
  data transfer over channel with, 209–214
  data transfer over lossy channel with, 214–217
  undetected, 445
bit-level error detection and correction, 444
BITNET, 62
BitTorrent, 141, 144–147, 158, 700
  trackers, 144–146
  trading algorithm, 146
blades, 495
block ciphers, 600–602
Bluetooth, 549–550
Boggs, David, 475, 478
Border Gateway Protocol (BGP), 374, 379, 386, 395–407, 504
  determining best routes, 398–402
  in Google SDN, 413
  hot potato routing, 400–401
  internal BGP, 397–398
  IP-anycast implementation with, 402–403
  outside-AS destinations, 400
  role of, 395–396
  route attributes, 399
  route information advertisement, 396–398
  route-selection algorithm, 401–402
  routing policy, 403–406
  routing tables, 401–402
border routers, 394–395, 495
botnets, 56
bottleneck bandwidth, 111
bottleneck link, 45
  TCP fairness and, 279–280
bounded delay, 311
bring home, 150
broadband Internet, 64
broadcast
  in ALOHA, 62, 458, 459, 460
  ARP messages, 471–473, 503
  CSMA and, 460–461
  CSMA/CD and, 462
  CTS and RTS frames, 541
  DHCP requests, 501–502
  Ethernet links, 480

broadcast. *(continued)*
forwarding to, 358
link-layer, 451
link-state, 379, 391
MAC address for, 470, 472
in OSPF, 393–394
packet sniffing and, 58
probe frames, 536
in switch poisoning, 485
wireless LANs, 451
broadcast address
IP, 341, 343
MAC, 470, 472
broadcast channels, 453
broadcast link, 451–453
broadcast media, 98, 303
live streaming as, 681
broadcast storms, 486–487
Brooks, Fred, 673
browsers, 681
brute-force attacks, 601
BS. *See* base station
BSC. *See* base station controller
BSS. *See* base station subsystem; basic service set
BTS. *See* base transceiver station
buffered distributors, 480
buffer overflows, congestion causing, 266–267
buffers
client application, 685–686
client-side, 682
output, 24
receive, 253, 254
send, 235
sizing for routers, 325
in streaming, 685–686
TCP, 685–686
buffer starvation, 689
bus, switching via, 320
Bush, Vannevar, 63

C

CA. *See* Certification Authority
cable Internet access, 14–15, 64
binary exponential backoff in, 463
DOCSIS, 465–467
cable modem termination system (CMTS), 15, 465–466
caching, 303
DNS, 134–135
push, 155–156
Web, 110–116
Caesar cipher, 597
call admission, 724, 725
call routing, to mobile users, 576–577
call setup protocol, 725
call setup signaling, 725
canonical hostname, 128
care-of address (COA), 564
in agent discovery, 573
indirect routing and, 565–567
carrier sense multiple access (CSMA), 455, 459–461
carrier sensing, 459
carrier-sensing random access, 537
CAST, 630
CBC. *See* Cipher Block Chaining
CD. *See* compact disk
CDMA. *See* code division multiple access
CDNs. *See* Content Distribution Networks
cells, 553
cell towers, 522
cellular data networks, 523
CDMA in, 528
cellular Internet access, 551
cellular network architecture, 551–554
4G, 557–560
3G, 554–557
2G, 553–554

cellular networks
CDMA use in, 455
4G, 520, 552, 553, 557–560
GSM, 551–554, 577–580
LTE, 18, 552, 553, 557–560
mobility management in, 574–582
3G, 18, 520, 523, 554–557, 677
2G, 553–554
cellular telephony, 18, 64, 455, 519
centralized designs, 129–130
centralized routing algorithm, 378–379
in LS algorithm, 380
central office (CO), 13–14
Cerf, Vinton, 62, 234, 371–372, 516
CERT Coordination Center, 596
certificates, 620
Certification Authority (CA), 619–620, 630
channel partitioning protocols, 453–455, 528, 537
CDMA, 457
FDM, 457
TDM, 456–457
channel propagation delay, 461
channels
with bit errors, 210–216
broadcast, 453
802.11, 534–537
lossy, 214–217
multiple access, 451, 452
perfectly reliable, 208, 209
satellite radio, 21
terrestrial radio, 20–21
channel utilization, 219
Check Point, 652, 660
checksum field, 236
checksumming, 448
checksums
corrupted ACK and NAK packet detection, 212
error detection with, 448
hash functions and, 611–612
Internet, 448, 611–612
IPv4 headers, 331–332
UDP, 204–206
China Telecom, 375
China Unicom, 375
chipping rate, 528
choke packets, 268
chosen-plaintext attacks, 599
chunks
BitTorrent, 144–146
Netflix streaming platform, 155
CIDR. *See* Classless Interdomain Routing
Cipher Block Chaining (CBC), 602–604, 641
in SSL, 636
ciphertext, 597
ciphertext-only attacks, 599
circuit, 27
circuit switching, 27–31
packet switching *versus*, 30–31
Cisco, 4, 64, 652, 660
Cisco 12000 series, switching fabric, 320–321
Cisco Catalyst 6500 Series, 318
switching bus, 320
Cisco Catalyst 7600 Series, 318
switching fabric, 321
Cisco Catalyst 8500 Series, switching fabric, 320
Cisco CRS, switching strategy, 321
Clark, Jim, 63, 184
classful addressing, 338–339
Classless Interdomain Routing (CIDR), 337–338, 502
cleartext, 597
Clear to Send (CTS), 540–542
client application buffers, 685–686
client buffering, 682

client process, 233
clients, 11, 86
  processes, 88–89
client-server architecture, 86, 87
  file distribution, 141–144
  TCP socket programming, 165
  UDP socket programming, 160
client-side buffers, 682
cloud computing, 65
  Amazon, 154–156
cloud data centers, 496
  hierarchical architecture in, 497–498
cloud services, response time of, 275
cluster selection strategy, 153
CMTS. *See* cable modem termination system
CNAME records, 136
CO. *See* central office
COA. *See* care-of address
coaxial cable, 20
code division multiple access (CDMA), 455, 457, 520, 528–532
collide, 453
collision avoidance, 537
  RTS/CTS frames for, 540–542
collision detection, 459
  CSMA/CD, 459, 461–464, 532, 539
  802.11 MAC protocol and, 537–538
collisions
  in broadcast channels, 453
  in CSMA, 460–461
  CSMA/CD and, 539
  FDM avoiding, 455
  link-layer switching eliminating, 484
  random access protocols and, 455
  in slotted ALOHA, 456
  TDM eliminating, 454
communication layer, SDN, 410
communication links, 4
Communications-Electronics Security Group, 605
compact disk (CD), 678
computational complexity, of LS algorithm, 382
computer networks, 2
  graph model of, 377–378
  history of, 59–65
  process interface to, 89
  throughput in, 43–46
conditional GET, 114–116
confidentiality, 594, 627
congestion
  alternative algorithms, 277
  buffer overflows from, 266–267
  causes and costs of, 261–267
  delays from, 263
  lost segments and, 271
  multihop paths and, 265–267
  retransmission and, 264–265
  routers and, 262–267
  throughput and, 262–267
congestion avoidance, 273–274
congestion control, 192, 253
  ABR, 203
  adaptive, 203
  AIMD, 277
  approaches to, 268
  bandwidth and, 271
  end-to-end, 268
  network-assisted, 268, 269
  principles of, 261–268
  TCP, 269–283
congestion window, 270, 276
connection flooding, 57
connectionless demultiplexing, 195–196

connectionless multiplexing, 195–196
connectionless transport, 200–206
connection management, TCP, 255–259, 261
connection-oriented demultiplexing, 196–199
connection-oriented multiplexing, 196–199
connection-oriented transport, 233–261
connection replay attack, 636
connection requests, 197
connection state, 202
connection tables, 655
Content Distribution Networks (CDNs), 114, 149–153
  case studies, 153–157
  cluster selection strategy, 153
  data centers, 150
  DNS and, 151–152
  Google, 156
  IP-anycast and, 402
  ISPs and, 150
  IXPs and, 150
  Kankan, 156–157
  Netflix, 154–156
  operation of, 150–153
  private, 150, 154–156
  third-party, 150
  video streaming and, 152–153
  YouTube, 156
content ingestion, 154
content processing, 154
content provider networks, 34
control frames, 540–542
control packets, 314
  in Skype, 697
control plane, 305, 315, 373
  4G, 558
  SDN, 407–416
convergence, routing algorithm speed of, 391
conversational voice and video, 680–681
cookies, 108–110
  SYN, 260
correspondent, 562
  in indirect routing, 566–567
correspondent agent, 568
countdown timer, 216
CRC. *See* cyclic redundancy check
crossbar switches, 319–321
cryptographic hash functions, 611–613
cryptography, 674
  attack types against, 599
  components, 597
  principles of, 596–610
  public key encryption, 597, 604–610, 630, 636
  symmetric key, 598–604, 625, 627, 630, 636
CSMA. *See* carrier sense multiple access
CSMA/CA. *See* CSMA with collision avoidance
CSMA/CD. *See* CSMA with collision detection
CSMA with collision avoidance (CSMA/CA), 537, 539
CSMA with collision detection (CSMA/CD), 459, 461–464, 532, 539
CSNET, 62
CTS. *See* Clear to Send
cumulative acknowledgement, 224, 238
customer, 32

cwnd, 270, 272–276
Cyclades, 61
cyclic redundancy check (CRC), 449–451, 526
  in 802.11, 538, 543
  in Ethernet frames, 477

D

DARPA. *See* Defense Advanced Research Projects Agency
DARTnet, 736
DASH. *See* Dynamic Adaptive Streaming over HTTP
data center network design, 495
data center networks, 495–500
  hierarchical architecture, 497–498
  load balancing, 496–497
  trends in, 498–500
data centers, 11, 86
  CDNs, 150
  Google, 151
  hosts, 495
  modular, 498–499
Data Center TCP (DCTCP), 283, 285
DATA frames, 540–542
Datagram Congestion Control Protocol (DCCP), 285–286
datagrams, 51, 191
  indirect routing of, 571
  inspecting, 348
  IP, 501
  IPsec, 641–644
  IPv4 format, 330–332
  IPv4 fragmentation, 332–335
  IPv6 format, 349–351
  NAT and, 346–347
  network-layer, 54
  reassembly of, 333–334, 351
  subnet dispatch of, 473–474
Data-Over-Cable Service Interface Specifications (DOCSIS), 465–467
  binary exponential backoff in, 463
data plane, 305, 361
  4G, 558
  generalized forwarding and SDN, 354–361
  IP, 329–354
  routers, 313–329
  SDN and, 408, 414–415
data received events, 245, 246
Davies, Donald, 60
DCCP. *See* Datagram Congestion Control Protocol
DCTCP. *See* Data Center TCP
DDoS. *See* distributed DoS
decentralized routing algorithm, 378–379
DECnet, 470
decryption algorithm, 597
deep packet inspection (DPI), 348, 354, 659
Deering, Steve, 589
Defense Advanced Research Projects Agency (DARPA), 61, 62, 371
delays
  adaptive playout, 692–694
  application, 43
  bounded, 311
  channel propagation, 461
  in end systems, 43
  end-to-end, 41–43, 689, 690
  fixed playout, 691–692
  network congestion and, 263
  nodal, 36
  nodal processing, 35
  in packet-switched networks, 35–46
  playout, 691–692, 692–694
  processing, 36

propagation, 35, 37–39
queuing, 24–25, 35, 36, 39–41, 263
in shared medium, 43
total nodal, 35
transmission, 35, 36–39
types of, 35–39
delay-sensitive applications, 680
demilitarized zone (DMZ), 661
demultiplexing, 193–200, 502
connectionless, 195–196
connection-oriented, 196–199
transport-layer, 192
denial-of-service (DoS) attacks, 56–57
distributed, 57, 58, 139
SYN floods for, 260
DES (Data Encryption Standard), 602, 630
destination-based forwarding, 315, 316–319
destination port number, 236
destination port number field, 194
DHCP. *See* Dynamic Host Configuration Protocol
DHCP ACK message, 344–345, 502
DHCP discover message, 343
DHCP offer message, 343–344
DHCP request message, 344, 501
dial-up Internet access, 16, 491
DIAMETER, 537, 650
DIC, 277
differentiated service, 709
Diffserv, 719–723
Diffie-Hellman Key Exchange, 604
Diffserv, 719–723
DIFS. *See* Distributed Inter-frame Space
Digital Attack Map, 56
digital signatures, 610, 614–620
digital subscriber line (DSL), 13–14, 64
digital subscriber line access multiplexer (DSLAM), 13–14
Dijkstra's algorithm, 379, 386
in OSPF, 392
directional antennas, 542
direct routing, 568–569
Direct Sequence Wideband CDMA (DS-WCDMA), 556
distance-vector algorithm (DV algorithm), 384–391
decentralization, 386
link-cost changes and link failure, 388–390
LS compared with, 390–391
message complexity, 390–391
poisoned reverse, 390
robustness, 391
speed of convergence, 391
distributed applications, 5
distributed DoS (DDoS), 57
DNS servers targeted by, 139
Distributed Inter-frame Space (DIFS), 539
distribution time, 141–144
DMZ. *See* demilitarized zone
DNS. *See* domain name system
DNS caching, 134–135
DNS query message, 503
DNS reply message, 504
DNS servers, 127
authoritative, 132, 504
BIND, 127
DDoS attacks targeting, 139
interactions of, 133
local, 132
root, 131, 134
TLD, 130, 131
DOCSIS. *See* Data-Over-Cable Service Interface Specifications
DOCSIS 2.0, 15
domain names, 406

domain name system (DNS), 50, 126–140, 503
CDNs and, 151–152
distributed, hierarchical database, 130–134
inserting records, 138, 140
Internet presence and, 406–407
intra-domain routing to, 503–504
intruder interference with, 596
IP-anycast in, 402
iterative queries, 133
messages, 136–138, 503–504
operation of, 129–135
query chain, 133–134
records, 135–136, 138, 140
recursive queries, 133–134
resource records, 135–136, 504
services provided by, 127–129
UDP usage by, 201
vulnerabilities, 139
DoS. *See* denial-of-service attacks
dotted-decimal notation, 335, 470–471
DPI. *See* deep packet inspection
dropping
OpenFlow, 358
packets, strategies for, 324
drop-tail, 324
DSL. *See* digital subscriber line
DSLAM. *See* digital subscriber line access multiplexer
duplicate ACKs, 214, 249
duplicate data packets, 216
duplicate packets, 212
DV algorithm. *See* distance-vector algorithm
Dynamic Adaptive Streaming over HTTP (DASH), 148–149, 155, 688
Dynamic Host Configuration Protocol (DHCP), 342–344
address obtainment with, 342–345
messages, 343–344, 501–502
mobile nodes and, 344
NAT and, 345
requests, 501–502
dynamic routing algorithms, 379
DYSWIS, 736

E

EAP. *See* Extensible Authentication Protocol
EAPoL, 649
Earthlink, 523
eavesdropping, 596
e-Bay, 64
eBGP. *See* external BGP
EC2, 65
ECE. *See* Explicit Congestion Notification Echo
echo request, 419
ECN. *See* Explicit Congestion Notification
edge routers, 314
Educause, 131
efficiency
of CSMA/CD, 464
of slotted ALOHA, 457–458
of slotted multiple access protocols, 457
802.11. *See* IEEE 802.11
EIGRP protocol, 392
elastic applications, 92
e-mail, 116–126
access protocols, 122–126
IMAP, 123, 125–126
message formats, 121–122
securing, 626–631
servers, 116–117, 128
SMTP, 50, 97, 118–123
web-based, 126
encapsulation, 53–55
in indirect routing, 566

Encapsulation Security Payload (ESP), 640, 642–643
encrypted, 594
encryption
  attack types against, 599
  passwords, 623–624
  polyalphabetic, 599–600
  public key, 597, 604–610, 630, 636
  security associations and, 641
  standards for, 602
  symmetric key, 598–604
encryption algorithm, 597
end-end principle, 205
end-point authentication, 58–59, 595
end systems, 2, 4, 9, 11
  delay in, 43
end-to-end congestion control, 268
end-to-end connection, 27
end-to-end delay, 41–43, 689, 690
enhanced packet core (EPC), 558
eNodeB, 558
enter deep, 150, 151
entity body, 106
EPC. *See* enhanced packet core
error checking, UDP checksums and, 204–206
error correction, 443
  bit-level, 444
  techniques for, 444–451
error detection, 210, 443
  bit-level, 444
  checksumming, 448
  parity bits, 446–448
  techniques for, 444–451
ESP. *See* Encapsulation Security Payload
EstimatedRTT, 242
Estrin, Deborah, 589–591
Ethane project, 415–416
Ethernet, 5, 16–17, 443, 460, 474–475
  binary exponential backoff in, 463
  broadcast, 451
  CSMA used by, 455
  development of, 62
  802.1Q-tagged VLAN frames, 490
  frames, 501
  frame structure, 476–478, 490
  Gigabit, 480
  MAC protocol, 479
  MTU, 235
  packet sniffing, 58
  repeater, 479
  run lengths, 479–480
  speeds, 479–480
  technologies, 478–481
  topologies, 475
even parity schemes, 446
event-based programming, 225
EWMA. *See* exponential weighted moving average
expedited forwarding PHB, 722
Explicit Congestion Notification (ECN), 282–283
Explicit Congestion Notification Echo (ECE), 283
exponential weighted moving average (EWMA), 242
extended FSM, 224
Extensible Authentication Protocol (EAP), 649
external BGP (eBGP), 397

F

Facebook, 676
fading, 528
failover paths, 494
fairness
  of AIMD, 279–282
  parallel TCP connections and, 282
  TCP and, 279–282
  UDP and, 281–282
fast recovery, 274–276

fast retransmit, 249–251
FCFS. *See* first-come-first-served
FDDI. *See* fiber distributed data interface
FDM. *See* frequency-division multiplexing
FDMA, 553
feature phones, 590
FEC. *See* forward error correction
Feynman, Richard, 304
FHSS. *See* frequency-hopping spread spectrum
fiber distributed data interface (FDDI), 465, 475
fiber optics, 64
  in cable systems, 14–15
  physical media, 20
fiber to the home (FTTH), 15–16, 64
FIFO. *See* first-in-first-out
file distribution
  client-server, 141–144
  P2P, 140–147
filtering, 481
FIN bit, 237
finite-state machine (FSM)
  for data transfer over channel with bit errors, 210–216
  for data transfer over lossy channel with bit errors, 216–217
  for data transfer over perfectly reliable channel, 208, 209
  extended, 224
  for GBN protocol, 222–224
  TCP congestion control, 273, 274
firewalls, 348, 354, 595, 651–659
  application gateways, 656–657, 659
  stateful filters, 652, 654–656
  traditional packet filters, 652–654
first-come-first-served (FCFS), 325
first-in-first-out (FIFO), 325–326
fixed playout delay, 691–692
flag days, 352
flag field, 236
flow, 349
flow-based forwarding, 407–408
flow control, TCP, 252–254
flow-control service, 252
flow table, 356
  match-plus-action, 415
  SDN, 410
  wildcards in, 357
Floyd, Sally, 437–438
foreign address, 564
foreign agent, 562
  anchor, 569
  discovery, 570, 571–572
foreign network, 562–564
forward error correction (FEC), 448, 689
  packet loss recovery with, 694–695
  in Skype, 697
forwarding, 306, 309, 312
  to broadcast, 358
  destination-based, 315, 316–319
  flow-based, 407–408
  generalized, 315, 354–361
  link-layer switches, 481–482
  longest prefix matching rule, 317, 339
  MPLS-enhanced, 493
  OpenFlow, 358
  packets, 308
  SDN, 407–408
forwarding plane, 314–315
forwarding tables, 25–26, 308, 309
  in input processing, 317–318
  IP, 502
  line cards, 317
  in LS algorithm, 381–382
  match-plus-action, 356
  prefixes, 317

routers, 308, 309
in SDN, 314, 316
4G, 520, 552
core network, 557–559
data plane, 558
network architecture, 557–560
QoS in, 558
radio access network, 559–560
tunneling in, 558
wireless LANs *versus*, 552
4G LTE, 552, 553
fragment, 333
fragmentation
IPv4 datagram, 332–335
IPv6 datagram, 351
frame control fields, 545
frame relay, 492
frames, 52
ACK, 541–542
acknowledgement, 539
beacon, 535, 536, 548
collisions between, 453
control, 540–542
CTS, 540–542
DATA, 540–542
802.11, 542–545
802.11 transmission, 538
Ethernet, 501
Ethernet structure for, 476–478, 479
link-layer, 54, 440
MPLS, 492–493
probe, 536
RTS, 540–542
time, 454
VLAN, 490
framing, 442
frequency-division multiplexing (FDM), 28–29, 453, 554
channel partitioning, 457
collision avoidance, 455
in DOCSIS, 465
orthogonal, 559
frequency-hopping spread spectrum (FHSS), 549
FSM. *See* finite-state machine
FTP protocol, 50
FTTH. *See* fiber to the home
full-duplex service, 233
fully connected topology, 498

G

Gateway GPRS Support Nodes (GGSNs), 556
Gateway Mobile services Switching Center (GMSC), 575
gateway router, 396
gateways, 315
application, 652, 656–657, 659
P-GW, 558
S-GW, 558
GBN protocol. *See* Go-Back-N (GBN) protocol
GE Information Services, 61
generalized forwarding, 315, 354–361
action, 358
match, 356–358
match-plus-action, 358–361
Generalized Packet Radio Service (GPRS), 556
generator, 449
geographically closest, 153
geostationary satellites, 21
GET requests, 104, 504
conditional, 114–116
DASH and, 148–149
HTTP streaming and, 148
GGSNs. *See* Gateway GPRS Support Nodes
Gigabit Ethernet, 480
GIST, 736

Global System for Mobile Communications (GSM), 551–552
handoffs in, 577–580
mobile IP commonalities to, 580
2G standards, 553–554
GMSC. *See* Gateway Mobile services Switching Center
Go-Back-N (GBN) protocol, 221–226
events, 224
TCP as, 252
Google, 11, 64, 285
CDNs, 156
data centers, 151
network infrastructure, 151
private network, 34, 65, 375
SDN use by, 375, 413, 416
web-based e-mail, 126
Google Application Engine, 65
Google Chrome browser, 158
QUIC protocol, 202, 203, 285
Google Chromium, 285
Google Talk, 675, 680, 699
GPRS. *See* Generalized Packet Radio Service
graph, 377
graph algorithms, 379
GSM. *See* Global System for Mobile Communications
guaranteed delivery, 311
guaranteed delivery with bounded delay, 311
guaranteed minimal bandwidth, 311–312
guided media, 19

H

H.263, 700
Handley, Mark, 589
handoff
in 802.11 subnets, 546–547
in GSM, 577–580
in wireless networks, 524
handshaking, 93
SSL, 633, 636–637
TCP three-way, 102, 165, 234, 256–257, 504
hard guarantees, 710
hash functions
checksums and, 611–612
cryptographic, 611–613
digital signatures using, 616
MD5, 612, 614
SHA-1, 613
HDLC. *See* high-level data link control
header length field, 236
header lines, 104, 106
headers
IPv4, 331–332
MIME, 630
MPLS, 492–493
RTP, 702
Web browsers and, 107–108
head-of-the-line blocking (HOL blocking), 322
HELLO message, 394
HFC. *See* hybrid fiber coax
hidden terminal problem, 528, 540–542
hierarchical architectures
within ASs, 394–395
in data center networks, 497–498
DNS distributed database, 130–134
Skype peers, 698
high bit rate, 676
high-level data link control (HDLC), 451
HLR. *See* home location register
HMAC, 614

HOL blocking. *See* head-of-the-line blocking
home agent, 562
  discovery, 570, 571–572
  indirect routing and, 565–566
  registration with, 572–573, 574
home location register (HLR), 575
  call routing and, 576–577
home MSC, 575
home network, 562, 575
home public land mobile network (home PMLN), 575
Home Subscriber Service (HSS), 559
hop limit, 351
host addresses, obtaining with DHCP, 342–345
host aliasing, 128, 136
hostname, 126–127
  alias, 128, 136
  in DNS queries, 132–133
  in DNS resource records, 135–137
  DNS services and, 127–128
hosts, 2, 9, 11
  data center, 495
  wireless, 520
Hotmail, 126
hot potato routing, 400–401
HSPA (High Speed Packet Access), 557
HSS. *See* Home Subscriber Service
HTML, development of, 63
HTML file, 98
HTTP. *See* HyperText Transfer Protocol
HTTP byte-range header, 687
HTTP GET message, 504
HTTP protocol, 50, 63
HTTP request, 502
HTTP response message, 505
HTTP streaming, 148–149, 681
  adaptive, 681
  client application buffer, 685–686
  DASH, 148–149, 155, 688
  early termination, 687–688
  prefetching video, 684–685
  repositioning video, 687–688
  TCP buffer, 685–686
  YouTube use of, 156
hub, 475
HughesNet, 16
Hulu, 679
hybrid fiber coax (HFC), 14–15, 439
hyperlinks, 102
HyperText Transfer Protocol (HTTP), 96–97, 98–100
  cookies, 108–110
  ICMP and, 419
  message format, 103–108
  non-persistent connections, 100–103
  persistent connections, 100, 103
  ports, 199–200
  request message, 103–105
  response message, 105–108
  SMTP comparison with, 121

I

IANA, 349
iBGP. *See* internal BGP
IBM, 61
ICANN. *See* Internet Corporation for Assigned Names and Numbers
ICMP. *See* Internet Control Message Protocol
ICMP messages, 139
IDEA, 630
IDSs. *See* intrusion detection systems
IEEE 802.1Q, 489, 491
IEEE 802.3 (Ethernet), 479
IEEE 802.3z (Gigabit Ethernet), 479
IEEE 802.5, 465

IEEE 802.11, 17, 520
access points, 533–534
adapters, 533, 537
ad hoc networks, 534
architecture, 533–537
authentication, 536–537
basic service set, 533, 544, 546–547
channels and association, 534–537
collision detection, 537–538
CRCs, 538, 543
frame transmission, 538
frequency ranges, 532
handoff in subnets, 546–547
hidden terminals and, 540–542
MAC addresses in, 469
MAC protocol, 537–542
mobility on same IP subnet, 546–547
as point-to-point link, 542
power management, 548
public access, 64, 523
rate adaptation, 547–548
RTS/CTS control frames, 540–542
standards, 532
IEEE 802.11ac, 532
IEEE 802.11b, interference from other devices, 525
IEEE 802.11 frames, 542
address fields, 543–545
MAC addresses in, 543–545
payload and CRC fields, 543
sequence number, duration, and frame control fields, 545
IEEE 802.11g, 532
IEEE 802.11i, 648–650
IEEE 802.11n, 532
IEEE 802.15.1, 549–550
IEEE 802.15.4, 550–551
IEEE 802.16, 560
IEEE 802 LAN/MAN Standards Committee, 5
IETF. *See* Internet Engineering Task Force
IETF Mobile Ad Hoc Network working group, 562
IKE. *See* Internet Key Exchange
IKE SA, 645
IMAP. *See* Internet Mail Access Protocol
indirect routing, 565–568, 571
information propagation, 303
infrastructure mode, 522
infrastructure wireless LANs, 534
Initialization Vector (IV), 603, 636, 648–649
in-order packet delivery, 311
input port, 314
input port processing, 316–319
forwarding tables in, 317–318
input queuing, 322
instantaneous throughput, 44
integrity checks, 641
Intel, 478
Intel 710 adapter, 443
intelligent software agents, 80
inter-area routing, 394–395
inter-autonomous system routing protocol, 395, 405
interconnection networks, 495
routing algorithms in, 499
switching via, 320–321
interface, 335
API, 89
NIC, 443
SDN controller, 410–411
socket, 6, 89
interference, 525
interleaving, 694, 696
internal BGP (iBGP), 397–398
internal router, 396

International Organization for Standardization (ISO), 52
International Telecommunication Union (ITU), 620
Internet. *See also* access networks
best-effort service in, 312
broadband, 64
Cerf on, 371–372
commercialization of, 63
components of, 2–5
DNS and presence on, 406–407
enterprise access, 16–17
history of, 59–65
home access, 13–16
network core, 21, 22
network edges, 9–11
network layer, 312
obtaining presence on, 406–407
registries, 341
router self-synchronization, 383
routing algorithms used in, 379
as service infrastructure, 5–6
services not provided by, 95
topology of, 486
transport layer, 191–193
transport services provided by, 93–95
Internet applications, transport protocols used by, 203
Internet checksum, 448, 611–612
Internet Control Message Protocol (ICMP), 419–421
IPv6 and, 421
message types, 420
packet filtering and, 653
Internet Corporation for Assigned Names and Numbers (ICANN), 138, 341, 392
Internet Engineering Task Force (IETF), 5, 348, 620, 736
Internet Exchange Points (IXPs), 33–34
CDNs in, 150
Google infrastructure at, 151, 156
Netflix infrastructure at, 155
Internet Key Exchange (IKE), 645
Internet Mail Access Protocol (IMAP), 123, 125–126
Internet of Things (IoT), 11, 590
Internet Protocol (IP), 5, 51, 371, 517. *See also* IPv4; IPv6
best-effort service, 688–689
ICMP and, 419
service model, 192
stack for, 50
total annual traffic using, 4
transition to, 62
Internet Real-Time Lab, 736
Internet registrars, 406
Internet Service Providers (ISPs), 4–5
access, 32
backbone, 404
CDNs and, 150
AS configurations, 392
global transit, 32
Google infrastructure at, 151, 156
multi-home, 33
multi-homed access, 404
Netflix infrastructure at, 155
peering agreements among, 404–405
PoP, 33
routing among, 395–407
Internet standards, 5
Internet Systems Consortium, 345
Internet telephony, 95, 680, 690
Internet video, 147–148
internetworking, 60–62
intra-autonomous system routing, 391–395, 405
SDN in, 416

intra-domain routing, 503–504
intrusion detection systems (IDSs), 348, 595, 659–662
intrusion prevention systems (IPSs), 348, 659
Intserv, 312
IoT. *See* Internet of Things
IP. *See* Internet Protocol
IP addresses, 63, 90, 127
  broadcast, 341, 343
  classes of, 338–339
  DHCP, 342–345
  Internet presence and, 406
  IPv4, 334–345
  IPv6, 349
  MAC addresses and, 469
  NAT and, 345–347
  obtaining blocks of, 341
  SIP and, 704–706
  socket programming, 159
  temporary, 342
IP-anycast, 402–403
IP datagram, 501
IP forwarding table, 502
IP fragmentation
  IPv4, 332–335
  IPv6, 351
iPhones, 18
IPsec, 354, 617, 637, 638–639
  AH and ESP, 640
  datagram, 641–644
  key management, 645
  packet forms, 641–642
  security associations, 640–641, 645
IP spoofing, 58–59, 623
IPSs. *See* intrusion prevention systems
IP traffic, volume of, 4
IPv4
  addressing, 334–345
  datagram format, 330–332
  datagram fragmentation, 332–335
  transitioning to IPv6 from, 352–354
IPv6, 348
  adoption of, 352–353
  datagram format, 349–351
  ICMP, 421
  transitioning to, 352–354
  tunneling, 352–353
IPX, 386, 470
IS-IS, 392, 413, 504
ISO. *See* International Organization for Standardization
ISO IDRP, 386
ISPs. *See* Internet Service Providers
iterative queries, 133
ITU. *See* International Telecommunication Union
IV. *See* Initialization Vector
IXPs. *See* Internet Exchange Points

J

jabbering adapters, 484
Jacobson, Van, 303–304, 589
Java, client-server programming with, 159
Jet Propulsion Laboratory, 372
jitter
  packet, 690–691
  removing, for audio, 691–694
Juniper MX2020, 314
Juniper Networks Contrail, 416

K

Kahn, Robert, 371, 372, 516
  ARPAnet development and, 60–62
  TCP/IP creation and, 234
Kankan, 156–157, 675
Karels, Mike, 303

keys, 597
IPsec management of, 645
SSL, 634
Kleinrock, Leonard, 60, 79–81, 371, 516
known-plaintext attacks, 599

L

label-switched router, 493
Lam, Simon, 516–518
Lampson, Butler, 358
LAN. *See* local area network
LAN address, 468
LAN-on-motherboard configuration, 443
layer 4 switching, 315
layer 5 switching, 315
layered architectures, 47–53
encapsulation, 53–55
layers, 49
leaky bucket policer, 716–719
least-cost path, 378
Bellman-Ford equation for, 384–385
in LS algorithm, 380–382
LEO satellites. *See* low-earth orbiting satellites
Level 3 Communications, 5
CDN, 150
Licklider, J. C. R., 60
Limelight, 150
line cards, 443
forwarding tables in, 317
input and output ports, 314
processing on, 320
line speeds, queuing and, 321–322
link access, 442
link capacity
buffer sizing and, 325
network congestion and, 263
link failure
DV algorithm and, 388–390
precomputed failover paths for, 494
link layer, 51–52, 440, 441
broadcast, 451
implementation locations, 443–444
IP datagram size and, 333
network as, 491–494
services provided by, 442–443
link-layer acknowledgement, 537, 538, 539
link-layer addressing, 468–474
link-layer frame, 54, 440
link-layer switches, 4, 23, 313, 481–487
destination address lookup in, 318
forwarding and filtering, 481–482
properties of, 484
self-learning, 483–484
links, 440
wireless, 521
link-scheduling discipline, 716
link-state algorithms (LS algorithms), 378, 379–383, 386
centralized routing algorithm, 380
computational complexity of, 382
DV compared with, 390–391
forwarding tables, 381–382
message complexity, 390–391
oscillations in, 382–383
OSPF, 392
robustness, 391
speed of convergence, 391
steps of, 380–381
link-state broadcast, 379
erroneous, 391
link virtualization, 491–494
link weights, in OSPF, 393
live streaming, 681
load balancers, 354, 496–497

load distribution, 128
load-insensitive algorithms, 379
load-sensitive algorithm, 379
local area network (LAN), 16–17. *See also* virtual local area networks; wireless LANs
  sniffing, 485
  switched, 467–491
local DNS server, 132
local preference, 401
logical communication, 188
logically centralized control, 374–375, 376, 437
logically centralized routing controllers, 310
longest prefix matching rule, 317, 339
Long-Term Evolution (LTE), 18, 552, 553
  network architecture, 557–560
  time slots in, 559–560
lookup algorithms, 318
loss anticipation schemes, 694–697
loss recovery schemes
  error concealment, 696–697
  FEC, 694–695
  interleaving, 696
loss-tolerant applications, 91, 680
lossy channels, 214–217
LoST, 736
lost segments, 271
low-earth orbiting (LEO) satellites, 21
LS algorithms. *See* link-state algorithms
LTE. *See* Long-Term Evolution
LTE-Advanced, 560

M

MAC. *See* message authentication code
MAC addresses
  ARP and, 471–472
  in beacon frames, 535
  broadcast address, 470, 472
  802.11 access points, 533, 535
  802.11 frames, 543–545
  network adapters, 468–470
  wireless LAN authentication by, 536
MAC protocol. *See* medium access control protocol
mailbox, 116
mail servers, 116–117
  aliasing, 128
malware, 55–56
  self-replicating, 56
managed device, 422
managed objects, 422
Management Information Base (MIB), 422, 424
managing server, 422
MANETs. *See* mobile ad hoc networks
manifest file, 149
MAP message, 465
Master Key, 649
Master Secret (MS), 636
match-plus-action, 318
  forwarding table, 356
  in generalized forwarding, 354–355
  OpenFlow, 358–361
match-plus-action flow tables, 415
maximum segment size (MSS), 235, 279
  negotiating, 236
maximum transmission unit (MTU), 235, 333
MD4, 613
MD5 authentication, 394
MD5 hash algorithm, 612, 614
MDCs. *See* modular data centers

medium access control protocol (MAC protocol), 442
802.11, 537–542
Ethernet, 479
medium access protocol, 439
memory
access times, 318
bandwidth of, 319
switching via, 319–320
mesh networks, wireless, 524
message authentication code (MAC), 613–614
digital signatures and, 616–617
IPsec datagrams, 642
in SSL, 634–635
message integrity, 594–595, 610, 627, 628
message queue, 117
messages, 23, 50, 88
application-layer, 54
ARP, 471–473, 503
complexity in LS algorithms, 390–391
DHCP, 343–344, 501–502
DNS, 136–138, 503–504
e-mail formats, 121–122
HELLO, 394
HTTP format, 103–108
ICMP ping, 139
intruder actions on, 596
OpenFlow, 415
port-status, 415
SIP, 706
source quench, 419–420
Metcalfe, Bob, 460, 475, 478
metering function, 722
MIB. *See* Management Information Base
Microsoft, 64
private network, 65
Microsoft Research, 375
middleboxes, 347, 354
MIME headers, 630
MIMO. *See* multiple input multiple-output
Minitel, 63
MME. *See* Mobility Management Entity
mobile ad hoc networks (MANETs), 524–525, 562
mobile IP, 570–574
agent discovery, 571–572
GSM commonalities to, 580
registration with home agent, 572–573, 574
mobile nodes
addressing, 562–564
DHCP and, 344
direct routing to, 568–569
in foreign networks, 563–564
indirect routing to, 565–568
routing to, 564–569
mobile station roaming number (MSRN), 576
mobile switching center (MSC), 554, 556, 575
anchor, 579
handoffs and, 578–580
mobile-user location protocol, 568
mobility, 519–520, 590
addressing and, 562–564
cellular network management of, 574–582
degrees of, 560–561
handoff and, 524
higher-layer protocols and, 580–582
management, 560–569
node address and, 561
on same IP subnet, 546–547
within VLANs, 548
wired infrastructure and, 562

Mobility Management Entity (MME), 559
modify-field action, 358
modular data centers (MDCs), 498–499
modulation techniques
  dynamic selection of, 527–528
  PCM, 678, 700
  SNR and BER for, 526–528
monoalphabetic cipher, 597
Mosaic Communications, 63
MOSPF. *See* multicast OSPF
MP3. *See* MPEG 1 layer 3
MPEG, 700
MPEG 1 layer 3 (MP3), 678, 700
MPLS. *See* Multiprotocol Label Switching
MPLS-enhanced forwarding, 493
MS. *See* Master Secret
MSC. *See* mobile switching center
MSRN. *See* mobile station roaming number
MSS. *See* maximum segment size
MTU. *See* maximum transmission unit
multicast OSPF (MOSPF), 394
multicast routing, 589
  in OSPF, 394
multi-home, 33
multi-homed access ISP, 404
multi-hop, infrastructure-based wireless networks, 524
multi-hop, infrastructure-less wireless networks, 524–525
multihop path, 265–267
multi-hop wireless networks, point-to-point 802.11 links in, 542
multimedia applications, 737
  audio properties, 677–679
  conversational voice and video over IP, 680–681
  network support for, 709–726
  streaming live audio and video, 681
  streaming stored audio and video, 679–680
  TCP use by, 202
  types of, 679–681
  UDP use by, 202–203
  video properties, 676–677
multipath propagation, 525
multiple access channels, 451, 452
multiple access problem, 451
multiple access protocols, 452–453, 537
multiple input multiple-output (MIMO), 532
multiple same-cost paths, in OSPF, 394
multiple versions, 677
multiplexing, 193–200
  connectionless, 195–196
  connection-oriented, 196–199
  transport-layer, 192
Multiprotocol Label Switching (MPLS), 491, 492–494
MX records, 128, 136

N

NAK (negative acknowledgments), 210–214, 448
  corrupted, 212
NAT. *See* network address translation; network address translator
National Physical Laboratory, 60
NAT translation table, 346
NAT traversal, 347
NCP. *See* network-control protocol
NCS. *See* network control server
negative acknowledgments, 210
neighbor, 377
neighboring peers, 145–146

Nelson, Ted, 63
Netflix, 147, 675, 679
  CDNs, 154–156
netnews, 673
Netscape Communications, 63–64, 184, 631
network adapters, 443, 444
  ARP query and, 472
  CSMA/CD operation and, 462–463
  datagram transmission and, 471, 473–474
  802.11, 533, 537
  error detection in, 448
  Ethernet frames and, 476–478
  jabbering, 484
  LAN-on-motherboard configuration, 443
  layer independence and, 470
  MAC address assignment, 468–470
  MAC addresses, 468–470
  on separate cards, 443
network address translation (NAT), 345–347, 354
  Skype traversal of, 697–699
network address translator (NAT), 318
network applications, 83
  architectures, 86–88
  communication for, 85
  principles of, 84–97
  proprietary, 158
  service requirements, 93
  standards-based, 158
  transport services available to, 90–93
network-assisted congestion control, 268, 269
network control functions, in SDN, 408
network-control protocol (NCP), 60, 62
network control server (NCS), 413
network core, 21–22
  circuit switching, 27–31
  4G networks, 557–559
  network of networks, 31–34
  packet switching, 22–26, 30–31
  3G networks, 556
network dimensioning, 709, 711–712
network functions virtualization (NFV), 416
Network Information Base (NIB), 413
network infrastructure, wireless networks and, 522–525
network interface card (NIC), 443
network layer, 51. *See also* control plane; data plane
  best-effort service, 312
  forwarding and routing, 306–311
  security, 312, 637–645
  services, 311–312
  transport layer relationship to, 188–191
network-layer datagram, 54
network management, 421–426
  defining, 421
  framework for, 422–423
  intruder interference with, 596
  link-layer switching and, 484
network management agent, 422–423
network management protocol, 423
network of networks, 31–34, 62
network operations center (NOC), 422
network protocols, 8–9
networks. *See also* access networks; cellular networks; Internet; local area network; wireless networks
  ad hoc, 534
  attacks against, 55–59
  CDNs, 114, 149–157
  cellular, 18, 523, 528, 551–560, 574–582
  content provider, 34

networks. *(continued)*
data center, 495–500
edges, 9–11, 564
foreign, 562–564
home, 562, 575
home PMLN, 575
mobile ad hoc, 524–525, 562
multimedia application support, 709–726
multimedia support, 709–726
packet-radio, 61
packet-satellite, 60
personal area, 548–551
PMLN, 575
private, 34, 65, 345, 375, 638
programmable, 408
proliferation of, 62–63
proprietary, 60–62
provider, 404
radio access, 556–557, 559–560
telephone, 491
throughput in, 43–46
topology of switched, 486
virtual-circuit, 492
visited, 562, 575
VLANs, 487–491, 497, 548
VPNs, 494, 548, 637–639
wireless mesh, 524
WPANs, 549–550
network security, 594–596
network-service applications, 416
network service model, 311–312
NeVoT, 736
NEXT-HOP, 399–400, 401
NFV. *See* network functions virtualization
NIB. *See* Network Information Base
NIC. *See* network interface card
NIST, 352, 602
nmap, 198, 261
NOC. *See* network operations center
nodal delay, 36
nodal processing delay, 35
nodes, 440
nomadic computing, 80
non-blocking switches, 320
nonce, 625, 648
non-persistent connections, 100–103
non-preemptive priority queuing, 327
Novell IPX, 386
NOX controller, 412, 416
NSFNET, 62, 63
nslookup program, 138
NS records, 136
NTT, 5

O

object, 98
OC. *See* Optical Carrier standard
odd parity schemes, 446
OFA. *See* Open Flow Agent
OFC. *See* Open Flow Controller
OFDM. *See* orthogonal frequency division multiplexing
offered load, 264
OLT. *See* optical line terminator
one-bit parity, 446–447
ONIX SDN controller, 413
ONOS, 412, 416, 418–419
ONT. *See* optical network terminator
OpenDaylight, 412, 416–417
OpenDaylight Lithium, 416
OpenFlow, 410, 412–413, 414–415
action, 358
flow table, 356
match, 356–358
match-plus-action, 358–361
Open Flow Agent (OFA), 413
Open Flow Controller (OFC), 413
Open Shortest Path First (OSPF), 374, 379, 392–395, 504
authentication in, 394

broadcast in, 393–394
Dijkstra's algorithm, 392
link weights, 393
multicast, 394
security and, 394
subnets, 392
Open Systems Interconnection (OSI), 52
operational security, 595, 651–662
firewalls, 651–659
IDSs, 348, 595, 659–662
opportunistic scheduling, 560
Optical Carrier standard (OC), 20
optical line terminator (OLT), 16
optical network terminator (ONT), 15
optimistically choked peers, 146
options field, 236
orthogonal frequency division multiplexing (OFDM), 559
OSI. *See* Open Systems Interconnection
OSI reference model, 50, 52–53
OSPF. *See* Open Shortest Path First
out-of-order packets, 225
output buffer, 24
output port, 314
forwarding to, 318
output port processing, 321
output queue, 24
output queueing, 323–324
outside-AS destinations, 400
OVSDB, 417

P

P2P architecture, 86–88, 517
file distribution, 140–147
scalability of, 141–144
Skype use of, 697–699
P2P live streaming, 147
P2P streaming, 157
P2P video streaming, 681
packet classification, 720
Packet Data Network Gateway (P-GW), 558
packet-dropping strategies, 324
packet filtering, 652–654, 661
packet header overhead, 202
packet headers
MPLS, 492–493
routing and, 308, 309
packet jitter, 690–691
packet loss, 25, 41, 321
error concealment, 696–697
FEC for, 694–695
interleaving for, 696
recovery from, 694–697
VoIP and, 689–690
packet marking, 714
packet-marking strategies, 324
Packet Radio, 516
packet-radio networks, 61
packets, 4, 23
ARP, 472
choke, 268
control, 314, 697
deep inspection of, 348, 354, 659
duplicate, 212
duplicate data, 216
forwarding, 308
in-order delivery of, 311
IPsec forms, 641–642
out-of-order, 225
processing, 486
RTP, 701
Packet Satellite, 516
packet-satellite networks, 60
packet scheduler, 325
packet scheduling
FIFO, 325–326
priority queuing, 326–328
round robin, 328–329
WFQ, 328–329

packet sniffer, 58, 77
packet-switched networks, delays in, 35–46
packet switches, 4, 23, 313
packet switching, 23–26, 27, 79
  circuit switching *versus*, 30–31
  development of, 59–60
  store-and-forward, 23–24
paging, 554
pairwise communication, 303
Pairwise Master Key (PMK), 650
parallel TCP connections, fairness and, 282
parity bit, 446
parity checks, 446–448
passive optical networks (PONs), 15–16
passive scanning, 536
passwords, 623–624
path loss, 525
paths, 4
  failover, 494
  high-bandwidth, 278–279
  least-cost, 378, 380–382, 384–385
  multihop, 265–267
  multiple same-cost, 394
  shortest, 378
Paxos, 413
payload field, 54
  in 802.11 frames, 543
PCM. *See* pulse code modulation
PDUs. *See* protocol data units
peering agreements, 404–405
peers, 33, 86–87
  BitTorrent, 144–146
  neighboring, 145–146
  optimistically choked, 146
  P2P streaming, 157
  relay, 698–699
  Skype, 698–699
  unchoked, 146
peer-to-peer applications, 140–147
per-connection throughput, 262–263
per-hop behavior (PHB), 720, 722
permanent address, 564
per-router control, 374, 375
persistent connections, 100, 103
personal area networks, 548–551
PGP. *See* Pretty Good Privacy
P-GW. *See* Packet Data Network Gateway
PHB. *See* per-hop behavior
Photobell, 79
physical address, 468
physical layer, 52
physical media, 18–21
  coaxial cable, 20
  fiber optics, 20
  satellite radio, 21
  terrestrial radio, 20–21
  twisted-pair copper wire, 19–20
piconet, 549
piggybacked acknowledgments, 241
ping, 419
ping messages, 139
pipelined reliable data transfer protocols, 217, 219–221
pipelining, 221
  TCP, 243
Plain Old Telephone Service (POTS), 697
plaintext, 597, 599
playback attack, 624
playout delay
  adaptive, 692–694
  fixed, 691–692
plug-and-play, 342, 484
PMK. *See* Pairwise Master Key
PMLN. *See* public land mobile network
PMS. *See* Pre-Master Secret
points of presence (PoPs), 33

point-to-point connections, 233
point-to-point link, 451
802.11 as, 542
Point-to-Point Protocol (PPP), 440, 451
MTU, 235
poisoned reverse, 390
polling protocol, 464
polls, 464
polyalphabetic encryption, 599–600
polynomial codes, 449
PONs. *See* passive optical networks
PoPs. *See* points of presence
port numbers, 90, 159
NAT and, 345–347
socket, 195–196
well-known, 194
port scanning, 198
port-status message, 415
positive acknowledgments, 210
Post Office Protocol—Version 3 (POP3), 123–125
POTS. *See* Plain Old Telephone Service
Pouzin, Louis, 61
power management, 548
PPLive, 147
PPP. *See* Point-to-Point Protocol
ppstream, 147
prefetching video, 684–685
prefix, 317, 318, 338, 339
Pre-Master Secret (PMS), 636
Pretty Good Privacy (PGP), 626, 630–631
Prim's algorithm, 379
priority queueing, 325, 326–328
non-preemptive, 327
privacy, 658
VoIP and, 699
private CDNs, 150
Netflix, 154–156
private key, 605
private networks, 34, 65, 345, 375, 638
probe frames, 536
processes
addressing, 89–90
client, 88–89
communicating, 88–90
network interface, 89
server, 88–89, 233
transport layer protocols connecting, 188
processing delay, 36
programmable network, 408
propagation delay, 35, 37–39
proprietary networks, 60–62
protocol data units (PDUs), 424, 425
protocol layering, 49–50
protocols, 5. *See also specific protocols*
defining, 7–9
network, 8–9
routing, 25–26
protocol stack, 50
provider, 32
provider networks, 404
proxy server, 110, 658
SIP, 707
PSH bit, 237
public key, 605
certifying, 630
public key certification, 617–620, 630–631
public key encryption, 597, 604–610
in PGP, 630
in SSL, 636
public land mobile network (PMLN), 575
public WiFi, 64, 523
pull protocol, 121
pulse code modulation (PCM), 678, 700

pure ALOHA protocol, 458
push caching, 155–156
push protocol, 121
Python, 158
  port numbers, 195
  TCP connections, 166–169
  UDP connections, 161–164, 195

Q

Q2931b protocol, 725
QoS. *See* quality of service
QQ, 680
quality of service (QoS)
  call admission, 724
  in 4G, 558
  per-connection guarantees, 710, 723–726
  resource reservation, 725
  RTP and, 701
  traffic policing and, 717
quantization, 678
query
  ARP, 472, 503
  DNS chain, 133–134
  DNS message, 503
queueing delays, 24–25, 35, 36, 39–41
  network congestion and, 263
queuing
  FIFO, 325–326
  input, 322
  line speed and, 321–322
  non-preemptive priority, 327
  output, 323–324
  priority, 325, 326–328
  round-robin, 325, 328–329
  in routers, 321–325
  traffic load and, 322
  transmission rate and, 321–322
  WFQ, 328–329
  work-conserving, 327, 328
QUIC protocol, 202, 203, 285

R

RA. *See* router agent
radio access network
  4G, 559–560
  3G, 556–557
Radio Network Controller (RNC), 556, 558
RADIUS, 537, 649–650
Rand Institute, 60
random access protocols, 453, 455–464, 478, 537
random backoff, 539
Random Early Detection (RED), 324
rarest first, 146
rate adaptation, 547–548
RC4 stream cipher, 647
RCP. *See* Routing Control Platform
realm with private addresses, 345
real-time conversational applications. *See also* Voice-over-IP
  protocols for, 700–708
  RTP, 700–703
  SIP, 703–708, 737
real-time measurements, 153
Real-Time Streaming Protocol (RTSP), 683
Real-Time Transport Protocol (RTP), 683, 700–702
  audio and video payload types, 703
  packet header fields, 702
reassembly
  IPv4 datagram, 333–334
  IPv6 datagram, 351
receive buffer, 253, 254
receiver
  in CRC operation, 449
  in parity bit operation, 446–448
receiver authentication, 627
receiver feedback, 210
receive window, 236, 253, 254
recursive queries, 133

RED. *See* Random Early Detection
reduced-function devices, 550
regional ISP, 32–33
registrar, 138, 406
  SIP, 707
registration
  with home agent, 572–574
  in mobile IP, 574
registries, 341
relay peers, 698–699
relays, Skype, 698–699
reliable data transfer, 91, 192, 231–232
  over channel with bit errors, 209–214
  over lossy channel with bit errors, 214–217
  over perfectly reliable channel, 208–209
  principles of, 206–232
  service implementation for, 207, 208
  service model for, 206, 207
  TCP, 244–351
reliable data transfer protocol, 206
  building, 208–217
  pipelined, 217, 219–221
reliable data transfer service, 244
reliable delivery service, link-layer, 442
repeater, 479
request line, 104
request messages, HTTP, 103–105
requests for comments (RFCs), 5
  as protocol standards, 158
Request to Send (RTS), 540–542
resource records (RRs), 135–136, 504
resource reservation, 725
response messages, HTTP, 105–108
response time, cloud service performance, 275
retransmission, 210
  congestion and, 264–265
  CSMA/CA and, 539
  CSMA/CD and, 539
  duplicate packets from, 212
  fast, 249–251
  in random access protocols, 455
  sequence numbers for handling, 212–213
  in slotted ALOHA, 456
  TCP timeout interval for, 242–243
  TCP timer management for, 244–245
  time-based, 216–217
Rexford, Jennifer, 436–438
RFC 2616, 158
RFCs. *See* requests for comments
RIP, 379, 386, 504
Rivest, Ron, 605, 612
RNC. *See* Radio Network Controller
roaming number, 576
Roberts, Lawrence, 60, 516
robustness, LS and DV algorithms, 391
root DNS servers, 131, 134
round-robin queuing, 325, 328–329
round-trip time (RTT), 102
  buffer sizing and, 325
  TCP estimation for, 241–243
  TCP variable tracking, 269–270
route, 4
  BGP, 399
  BGP selection algorithm for, 401–402
route aggregation, 339
route information, advertising in BGP, 396–398
router agent (RA), 374
router discovery, 571

routers, 4, 23, 354
architecture of, 313
border, 394–395, 495
buffer sizing, 325
components of, 313–316
congestion and, 262–267
data plane, 313–329
destination-based forwarding, 315, 316–319
edge, 314
forwarding plane, 314–315
forwarding tables, 308, 309
gateway, 396
input port processing, 316–319
internal, 396
label-switched, 493
NAT-enabled, 345–347
output port processing, 321
per-router control, 374, 375
queuing in, 321–325
self-synchronization, 383
switches *versus*, 485–487
switching fabric, 319–321
route summarization, 339
routing, 308, 309
calls to mobile users, 576–577
direct, to mobile nodes, 568–569
hot potato, 400–401
indirect, in mobile IP, 571
indirect, to mobile nodes, 565–568
inter-area, 394–395
intra-ASs, 391–395, 405, 416
intra-domain, 503–504
intruder interference with, 596
among ISPs, 395–407
link weights in, 393
logically centralized, 310
to mobile nodes, 564–569
multicast, 394, 589
routing algorithms, 308, 309, 376–391
ARPAnet, 379, 386
centralized, 378–380
convergence speed, 391
decentralized, 378–379
distance-vector, 384–391
dynamic, 379
in interconnection networks, 499
link-state, 379–383
load sensitivity, 379
static, 379
routing controllers
logically centralized, 310
SDN and, 311
Routing Control Platform (RCP), 436
routing loop, 389
routing policy, BGP, 403–406
routing processor, 314
routing protocols, 25–26
routing tables, 386
BGP, 401–402
RRs. *See* resource records
RSA algorithm, 605–610, 630
RST bit, 236
RSVP protocol, 725
RTP. *See* Real-Time Transport Protocol
RTP header, 701
RTP packet, 701
RTP session, 701
RTS. *See* Request to Send
RTSP. *See* Real-Time Streaming Protocol
RTT. *See* round-trip time
rwnd, 269–270

S

SA. *See* security association
SAD. *See* Security Association Database
SAL. *See* Service Abstraction Layer

SampleRTT, 242
satellite Internet access, 16, 439
satellite radio channels, 21
Scantlebury, Roger, 60
scheduling algorithms, 560
Schulzrinne, Henning, 700, 736–738
SCTP. *See* Stream Control Transmission Protocol
SDN. *See* software-defined networking
SDN controller, 410–412, 437
secure communication, 594
secure e-mail, 627–630
Secure Hash Algorithm (SHA-1), 613, 614
Secure Sockets Layer (SSL), 94, 184, 516, 631–637, 658
  connection closure, 637
  data transfer, 634–635
  handshake, 633, 636–637
  key derivation, 634
security, 673–674
  datagram inspection, 348
  DNS vulnerabilities, 139
  e-mail, 626–631
  firewalls, 348, 354, 595, 651–659
  IDSs, 348, 595, 659–662
  network layer, 312, 637–645
  operational, 384, 595, 651–662
  OSPF and, 394
  switch poisoning, 485
  SYN flood attacks, 260
  transport protocol, 92–93
  wireless LANs, 646–650
security association (SA), 640–641, 645
Security Association Database (SAD), 641
Security Parameter Index (SPI), 641
Security Policy Database (SPD), 644
segment replay attack, 636
segments, 51, 188, 191
  acknowledged, 271
  lost, 271
  maximum size, 235, 236, 279
  TCP, 235
  TCP structure, 236–241
  TCP SYN, 504, 653
  transport-layer, 54
  UDP, 501
  UDP structure, 204
selective acknowledgment, 252
selective repeat (SR), 221, 226–232
  events and actions, 228
  operation of, 229
  TCP as, 252
  window size, 230, 231
self-clocking, 270
self-learning, 483–484, 502
self-replicating malware, 56
self-scalability, 87
self-synchronization, 383
send buffer, 235
sender authentication, 627, 628
senders
  in CRC operation, 449–450
  in parity bit operation, 446
sending rate, 264
sequence number, 212
  in 802.11 frames, 545
  in GBN protocol, 221–222
  jitter control with, 691
  in pipelined protocols, 221
  retransmission handling with, 212–213
  RTP, 702
  in SR protocol, 227, 230
  in SSL MAC calculation, 635
  TCP, 237–239
  for TCP segment, 238
  Telnet and, 239–241
sequence number field, 236

servers, 11, 86, 88
authoritative DNS, 132, 504
DNS, 127, 131–134, 132, 139
DNS root, 131, 134
enter-deep, 151
local DNS, 132
mail, 116–117, 128
managing, 422
network control, 413
processes, 88–89, 233
proxy,658,707, 110
TCP socket programming, 168–170
UDP socket programming, 163–164
user interaction with via cookies, 108–110
web, 63, 99, 199–200
Service Abstraction Layer (SAL), 416–417
service differentiation, 709, 719–723
Service Level Agreements (SLAs), 422
service model, 49
IP, 192
network, 311–312
reliable data transfer, 206, 207
services
DNS, 127–129
flow-control, 252
full-duplex, 233
layering, 49
link layer, 442–443
network layer, 311–312
TCP, 192
transport layer, 90–95
UDP, 95
unreliable, 192
Service Set Identifier (SSID), 534
in beacon frames, 535
Serving Gateway (S-GW), 558
Serving GPRS Support Nodes (SGSNs), 556
Session Initiation Protocol (SIP), 703–708, 737
addresses, 705–706
call to known IP address, 704–705
messages, 706
name translation and user location, 706–708
session keys, 609, 627
SGSNs. *See* Serving GPRS Support Nodes
S-GW. *See* Serving Gateway
SHA-1. *See* Secure Hash Algorithm
Shamir, Adi, 605
shared medium, 20
delays in, 43
shipping containers, 498–499
shortest path, 378
Short Inter-frame Spacing (SIFS), 538
SIFS. *See* Short Inter-frame Spacing
signal strength, 525
fading, 528
signal-to-noise ratio (SNR), 526–528
rate adaptation and, 547
signature-based systems, 661, 662
silent periods, 29
simple authentication, 394
Simple Mail Transfer Protocol (SMTP), 50, 97, 116, 118–120
HTTP comparison with, 121
mail access protocols and, 122–123
Simple Network Management Protocol (SNMP), 417, 424–426
single-hop, infrastructure-based wireless networks, 524
single-hop, infrastructure-less wireless networks, 524
SIP. *See* Session Initiation Protocol
SIP addresses, 705–706
SIP proxy, 707

SIP registrar, 707
Skype, 675, 680, 697–700
audio and video quality, 697
control packets in, 697
P2P techniques in, 697–699
peer hierarchy, 698
relay peers, 698–699
TCP use by, 697
UDP use by, 95, 697
Slammer worm, 198
SLAs. *See* Service Level Agreements
sliding-window protocol, 222
slotted ALOHA
backoff in, 516
collisions in, 456
efficiency of, 457–458
retransmission in, 456
slow start, 272–273
small office, home office (SOHO), subnets, 345
smart phones, 590
smart spaces, 80
SMI. *See* Structure of Management Information
SMTP. *See* Simple Mail Transfer Protocol
SNA, 61
sniffing, 58, 77, 485
SNMP. *See* Simple Network Management Protocol
Snort, 662
SNR. *See* signal-to-noise ratio
social networks, 65
socket interface, 6, 89
socket programming, 157–158
client-server architecture, 160
IP addresses, 159
port numbers, 159, 195–196
TCP, 164–170
UDP, 159–164
sockets, 193
port numbers, 195–196
simultaneous, 198
TCP, 502, 504
welcoming, 197
soft guarantees, 710
software agents, 80
software-defined networking (SDN), 306, 311, 436, 437, 590
architecture of, 408
control applications, 410–412
control plane, 315, 407–416
data plane, 408, 414–415
forwarding tables in, 314, 316
generalized forwarding and, 354–361
key characteristics of, 407–408
link state change in, 414–415
logically centralized control in, 374–375
packet forwarding and, 312
routing processor responsibilities in, 314
SOHO. *See* small office, home office
source port number, 236
source port number field, 194
source quench message, 419–420
spanning trees, 486
spatial redundancy, 677
SPD. *See* Security Policy Database
spectrum access rights, 523
SPI. *See* Security Parameter Index
split-connection approaches, 582
Spotify, 676
Sprint, 5
SR. *See* selective repeat
SRI. *See* Stanford Research Institute
SSID. *See* Service Set Identifier
SSL. *See* Secure Sockets Layer
SSL record, 635

SSRC. *See* synchronization source identifier
ssthresh, 273–276
Stanford Research Institute (SRI), 60, 79
StarBand, 16
stateful filters, 652, 654–656
stateless protocols, 100
state-management layer, SDN, 410
static routing algorithms, 379
status line, 106
stop-and-wait protocols, 211, 219, 220
store-and-forward transmission, 23–24
stream ciphers, 600, 647
Stream Control Transmission Protocol (SCTP), 285
streaming
  adaptive HTTP, 681
  CDNs and, 152–153
  DASH, 148–149, 155, 688
  HTTP, 148–149, 681, 685–688
  live, 681
  live video, 681
  Netflix platform, 154–156
  P2P, 157
  P2P live, 147
  P2P video, 681
  processing for, 154
  RTSP, 683
  stored audio and video, 679–680
  TCP buffers in, 685–686
  UDP, 681, 683
  video, 147–148, 152–156
streetlamp wireless hotspots, 523
Structure of Management Information (SMI), 422
subnet mask, 336
subnets, 335–339
  datagram transmission to, 473–474
  mobility on, 546–547
  obtaining blocks of IP addresses, 341
  in OSPF, 392
  SOHO, 345
successful slot, 457
super peers, 698
SWAN, 375
switch, 475
switched networks, topology of, 486
switches
  crossbar, 319–321
  forwarding and filtering by, 481–482
  layer 4, 315
  layer 5, 315
  link-layer, 4, 23, 313, 318, 481–487
  non-blocking, 320
  plug-and-play, 484
  properties of, 484
  routers *versus*, 485–487
  self-learning, 483–484
  top of rack, 495
  VLANs and, 488
switch filtering, 481–482
switch forwarding, 481–482
switching, 312
  in destination-based forwarding, 318
  techniques for, 319–321
switching fabric, 314
  bus, 320
  crossbar, 319–321
  interconnection network, 320–321
  memory, 319–320
  queuing and speed of, 321–322
switch poisoning, 485
switch table, 481
  poisoning, 485
symmetric key cryptography, 598–604
  block ciphers, 600–602

cipher-block chaining, 602–604
nonce use with, 625
in PGP, 630
polyalphabetic encryption, 599–600
secure e-mail using, 627
in SSL handshake, 636
SYNACK segment, 255, 259
SYN bit, 237
synchronization source identifier (SSRC), 702
SYN cookies, 260
SYN flood attack, 260

T

Tag Protocol Identifier (TPID), 489
taking-turns protocols, 453, 464–465, 537
TCAMs. *See* Ternary Content Addressable Memories
TCP. *See* Transmission Control Protocol
TCP ACK bits, 653–654
TCP congestion-control algorithm, 271–276
TCP connection, 93
TCP-Friendly Rate Control (TFRC), 285–286
TCP/IP, 5, 234
TCP Reno, 276, 277
TCP segments, 235
TCP services, 93–95
TCP socket, 502, 504
TCP splitting, 275
TCP states, 257–259
TCP SYN segment, 504, 653
TCP Tahoe, 276
TCP Vegas, 277
TDM. *See* time-division multiplexing
telco. *See* telephone company
Telenet, 61
telephone company (telco), 13
telephone networks, 491
Telnet, 120, 239–241, 623, 656–657
Temporal Key (TK), 650
temporal redundancy, 677
temporary IP addresses, 342
Ternary Content Addressable Memories (TCAMs), 318
terrestrial radio channels, 20–21
TFRC. *See* TCP-Friendly Rate Control
3rd Generation Partnership Project (3GPP), 555, 557
Third Generation Partnership Program, 353
third-party CDNs, 150
3Com, 478
3G, 18, 520, 523
core network, 556
network architecture, 554–557
radio access network, 556–557
video over, 677
3GPP. *See* 3rd Generation Partnership project
three-way handshake, 102, 165, 234, 256–257, 504
throughput, 43–46, 91–92
average, 44
congestion and, 262–267
instantaneous, 44
per-connection, 262–263
TCP, 278
of transport layer, 91–92
tier-1 ISPs, 32–33
time-based retransmission, 216–217
time-division multiplexing (TDM), 28–30, 453–457, 554, 556
time frames, 454

timeout events
in GBN protocol, 224
in SR protocol, 228
TCP, 242–243, 245, 246
timeout intervals
doubling, 247–249
TCP, 242–243, 247–249
time slots, 454
in LTE, 559–560
timestamps, 691, 702
time-to-live (TTL), 331
TK. *See* Temporal Key
TLD. *See* top-level domain
TLS. *See* Transport Layer Security
token, 465
token-passing protocol, 465
token ring protocol, 465, 475
Tomlinson, Ray, 60
top-down approach, 50
top-level domain (TLD), DNS servers, 130, 131
Top of Rack switch (TOR switch), 495
TOR, 658
torrents, 144–146
TOR switch. *See* Top of Rack switch
TOS. *See* type of service
total nodal delay, 35
TPID. *See* Tag Protocol Identifier
Traceroute, 42–43, 420–421
trackers, 144–146
traditional packet filters, 652–654
traffic classes, 714
isolating, 715–716
traffic conditioning, 720
traffic engineering, 393
MPLS and, 494
traffic intensity, 39
traffic isolation, 487–488, 714
traffic load
buffers and, 325
queuing and, 322
traffic policing, 715
leaky bucket, 716–719
traffic profiles, 721–722
Transmission Control Protocol (TCP), 5, 191. *See also* Secure Sockets Layer
ACK bit, 653–654
ACK generation recommendation, 250
acknowledgment number, 237–239
buffers in streaming, 685–686
closing connection, 256–257
congestion avoidance, 273–274
congestion-control algorithm, 271–276
congestion control in, 269–283
congestion window, 270, 276
connection, 233–236
connection management, 255–259, 261
connection requests, 197
cumulative acknowledgement, 238
demultiplexing, 196–199
development of, 62
establishing connection, 255–256
fairness and, 279–282
fast recovery, 274–276
fast retransmit, 249–251
flow control, 252–254
full-duplex service, 233
high-bandwidth paths and, 278–279
Internet checksum in, 448
multimedia applications using, 202
parallel browser connections, 101–102
parallel connection fairness, 282
pipelining, 243

point-to-point connections, 233
receive window, 253, 254
reliable data transfer, 244–351
retransmission timeout interval, 242–243
RTT estimation, 241–243
securing, 94
segment structure, 236–241
selective acknowledgment, 252
self-clocking, 270
sequence number, 237–239
services provided by, 192
simultaneous connection sockets, 198
Skype use of, 697
slow start, 272–273
SMTP using, 119
socket client, 166–168
socket programming, 158, 164–170
socket server, 168–170
steady-state behavior of, 278
three-way handshake, 102, 165, 234, 256–257, 504
throughput, 278
timeout events, 242–243, 245, 246
timeout intervals, 242–243, 247–249
timer management, 244–245
transition to, 62–63
variables, 269–270, 273, 276
Web servers and, 199–200
wireless networks and, 581
transmission delay, 35, 36–39
transmission power, 527
transmission rate, 4
BER and, 527
queuing and, 321–322
transparent, 481
transport layer, 51
application services, 90–93
fragment reassembly and, 334–335
in Internet, 191–193
network layer relationship to, 188–191
reliable data transfer and, 91
security, 92–93
throughput of, 91–92
timing guarantees, 92
transport-layer multiplexing and demultiplexing, 192
transport-layer protocols, 188
Transport Layer Security (TLS), 631
transport-layer segment, 54
transport mode, 641
transport services
application availability of, 90–93
Internet, 93–95
network application requirements, 93
triangle routing problem, 568
triple-DES, 630
3DES, 602, 641
TTL. *See* time-to-live
tunnel, 352
tunneling, 352
in 4G networks, 558
tunnel mode, 641
twisted-pair copper wire, 19–20
two-dimensional parity, 447
2G cellular networks, 553–554
Tymnet, 61
type numbers, 477
type of service (TOS), 331, 713, 714

U

ubiquitous WiFi, 523, 552
UCLA, 79, 371
UDP. *See* User Datagram Protocol
UDP segment, 501
UDP services, 95

UDP socket programming, 158, 159–164
client, 161–163
port numbers, 195–196
server, 163–164
UDP streaming, 681, 683
UMTS (Universal Mobile Telecommunications Service), 555, 556
unchoked peers, 146
undetected bit errors, 445
unguided media, 19
unidirectional data transfer, 208
unlicensed spectrum, 523
unreliable services, 192
unshielded twisted pair (UTP), 19
URG bit, 237
urgent data pointer field, 237
URLs, SIP, 705–706
user agents, 116
User Datagram Protocol (UDP), 191, 192, 200–206
advantages of, 201–202
checksum, 204–206
connectionless nature of, 201
DNS using, 201
fairness and, 281–282
Internet checksum in, 448
multimedia applications using, 202–203
multiplexing and demultiplexing, 195–196
reliability with, 203–204
RTP and, 700–701
segment structure, 204
Skype use of, 95, 697
in VoIP, 689
user state, cookies, 108–110
utilization, 219
UTP. *See* unshielded twisted pair

V

VANET. *See* vehicular ad hoc network
VC networks. *See* virtual-circuit networks
vehicular ad hoc network (VANET), 525
Verisign Global Registry Services, 131
video
properties of, 676–677
RTP payloads, 703
Skype quality adaptation for, 697
streaming, 147–148, 152–156, 679–681
video compression, 677
video conferencing, 700
video streaming, 147–148
CDNs and, 152–153
live, 681
Netflix platform, 154–156
P2P, 681
prefetching, 684–685
processing for, 154
repositioning, 687–688
stored video, 679–680
virtual-circuit networks (VC networks), 492
virtual local area networks (VLANs), 487–491
in data center networks, 497
mobility within, 548
virtual private networks (VPNs), 494, 637, 638–639
mobility within, 548
viruses, 56
visited network, 562, 575
visitor location register (VLR), 575
call routing and, 577
VLANs. *See* virtual local area networks

VLAN tags, 489, 490
VLAN trunking, 489, 490
VLR. *See* visitor location register
Voice-over-IP (VoIP), 43, 520, 680, 697–700, 736
  best-effort IP service limitations and, 688–689
  end-to-end delay, 690
  jitter removal, 691–694
  packet jitter, 690–691
  packet loss, 689–690
  privacy concerns, 699
  RTP, 700–703
  SIP, 703–708
VoIP. *See* Voice-over-IP
VPNs. *See* virtual private networks
vulnerability attacks, 56

W

web-based e-mail, 126
Web browsers, 63–64, 88–89, 99
  conditional GET and, 115–116
  cookies, 108–110
  email access via, 126
  GET requests, 104
  header lines from, 107–108
  parallel connections, 101–102, 282
  SSL support, 631
  web caches and, 110–113
Web caching, 110–116
web of trust, 631
Web page, 98
web page requests, 500–505
Web servers, 63, 99
  TCP and, 199–200
Wechat, 675, 699
weighted fair queuing (WFQ), 328–329, 716–719
welcoming socket, 197
well-known application protocols, 195
well-known port numbers, 194
WEP. *See* Wired Equivalent Privacy
WFQ. *See* weighted fair queuing
wide-area wireless Internet access, 18
WiFi, 4, 5, 443, 520, 532
  address fields, 543–545
  architecture, 533–537
  enterprise usage of, 16–17
  frames, 542–545
  link layer implementation, 443
  MAC addresses in, 543–545
  MAC protocol, 537–542
  mobility on same IP subnet, 546–547
  packet sniffing, 58
  payload and CRC fields, 543
  power management, 548
  public, 64, 523
  rate adaptation, 547–548
  sequence number, duration, and frame control fields, 545
  ubiquitous, 523, 552
  wide-area wireless *versus*, 18
WiFi jungle, 535
wildcards, in flow table entries, 357
WiMAX (World Interoperability for Microwave Access), 560, 736
window scaling factor, 236
window size, 222
  in SR, 230, 231
Wired Equivalent Privacy (WEP), 646–648
wireless communication links, 521, 525–528
  differences from wired links, 525
  dynamic selection of modulation techniques, 527–528
  interference, 525
  modulation techniques, 526–528
  multipath propagation, 525

wireless communication links. *(continued)*
signal strength, 525
transmission power, 527
transmission rate, 527
wireless hops, 524–525
wireless hosts, 520
wireless Internet devices, 519–520
wireless LANs, 17, 439
authentication, 536–537
broadcast, 451
CDMA in, 528
4G *versus*, 552
infrastructure, 534
securing, 646–650
wireless mesh networks, 524
wireless networks, 590
ad hoc, 534
elements of, 520–524
handoff in, 524
higher-layer protocols and, 580–582
infrastructure and, 522–523, 524–525
mobile ad hoc, 524–525, 562
packet sniffing, 58
types of, 524–525
vehicular ad hoc, 525
wireless personal area networks (WPANs), 549–550
Wireless Philadelphia, 523
Wireshark, 58, 77–78, 487
work-conserving queuing, 327, 328
World Wide Web, 83, 98
worms, 56, 198
WPANs. *See* wireless personal area networks

X

X.25 protocol suite, 63, 517
X.509, 620
Xerox Palo Alto Research Center (Xerox PARC), 478
XTP, 448

Y

Yahoo, 64
web-based e-mail, 126
Youku, 147, 675
YouTube, 147, 679
CDNs, 156
data centers, 151

Z

zeroconf, 342
Zigbee, 550–551
Zimmerman, Phil, 630