

wireless LAN technologies described in Section 1.2.1 use local-area radio channels; the cellular access technologies use wide-area radio channels. We'll discuss radio channels in detail in Chapter 7.

Satellite Radio Channels

A communication satellite links two or more Earth-based microwave transmitter/ receivers, known as ground stations. The satellite receives transmissions on one frequency band, regenerates the signal using a repeater (discussed below), and transmits the signal on another frequency. Two types of satellites are used in communications: geostationary satellites and low-earth orbiting (LEO) satellites [Wiki Satellite 2016]. Geostationary satellites permanently remain above the same spot on Earth. This stationary presence is achieved by placing the satellite in orbit at 36,000 kilometers above Earth's surface. This huge distance from ground station through satellite back to ground station introduces a substantial signal propagation delay of 280 milliseconds. Nevertheless, satellite links, which can operate at speeds of hundreds of Mbps, are often used in areas without access to DSL or cable-based Internet access. LEO satellites are placed much closer to Earth and do not remain permanently above one spot on Earth. They rotate around Earth (just as the Moon does) and may communicate with each other, as well as with ground stations. To provide continuous coverage to an area, many satellites need to be placed in orbit. There are currently many low-altitude communication systems in development. LEO satellite technology may be used for Internet access sometime in the future.

1.3 The Network Core

Having examined the Internet's edge, let us now delve more deeply inside the network core—the mesh of packet switches and links that interconnects the Internet's end systems. Figure 1.10 highlights the network core with thick, shaded lines.

1.3.1 Packet Switching

In a network application, end systems exchange messages with each other. Messages can contain anything the application designer wants. Messages may perform a control function (for example, the “Hi” messages in our handshaking example in Figure 1.2) or can contain data, such as an e-mail message, a JPEG image, or an MP3 audio file. To send a message from a source end system to a destination end system, the source breaks long messages into smaller chunks of data known as packets. Between source and destination, each packet travels through communication links and packet switches (for which there are two predominant types, routers and link-layer switches). Packets are transmitted over each communication link at a rate equal to the full transmission rate of the link. So, if a source end system or a packet switch is sending a packet of L bits over a link with transmission rate R bits/sec, then the time to transmit the packet is L/R seconds.

Store-and-Forward Transmission

Most packet switches use store-and-forward transmission at the inputs to the links. Store-and-forward transmission means that the packet switch must receive the entire packet before it can begin to transmit the first bit of the packet onto the outbound link. To explore store-and-forward transmission in more detail, consider a simple network consisting of two end systems connected by a single router, as shown in Figure 1.11. A router will typically have many incident links, since its job is to switch an incoming packet onto an outgoing link; in this simple example, the router has the rather simple task of transferring a packet from one (input) link to the only other attached link. In this example, the source has three packets, each consisting of L bits, to send to the destination. At the snapshot of time shown in Figure 1.11, the source has transmitted some of packet 1, and the front of packet 1 has already arrived at the router. Because the router employs store-and-forwarding, at this instant of time, the router cannot transmit the bits it has received; instead it must first buffer (i.e., “store”) the packet's bits. Only after the router has received all of the packet's bits can it begin to transmit (i.e., “forward”) the packet onto the outbound link. To gain some insight into store-and-forward transmission, let's now calculate the amount of time that elapses from when the source begins to send the packet until the destination has received the entire packet. (Here we will ignore propagation delay—the time it

takes for the bits to travel across the wire at near the speed of light—which will be discussed in Section 1.4.) The source begins to transmit at time 0; at time L/R seconds, the source has transmitted the entire packet, and the entire packet has been received and stored at the router (since there is no propagation delay). At time L/R seconds, since the router has just received the entire packet, it can begin to transmit the packet onto the outbound link towards the destination; at time $2L/R$, the router has transmitted the entire packet, and the entire packet has been received by the destination. Thus, the total delay is $2L/R$. If the Figure 1.11 Store-and-forward packet switching switch instead forwarded bits as soon as they arrive (without first receiving the entire packet), then the total delay would be L/R since bits are not held up at the router. But, as we will discuss in Section 1.4, routers need to receive, store, and process the entire packet before forwarding. Now let's calculate the amount of time that elapses from when the source begins to send the first packet until the destination has received all three packets. As before, at time L/R , the router begins to forward the first packet. But also at time L/R the source will begin to send the second packet, since it has just finished sending the entire first packet. Thus, at time $2L/R$, the destination has received the first packet and the router has received the second packet. Similarly, at time $3L/R$, the destination has received the first two packets and the router has received the third packet. Finally, at time $4L/R$ the destination has received all three packets! Let's now consider the general case of sending one packet from source to destination over a path consisting of N links each of rate R (thus, there are $N-1$ routers between source and destination). Applying the same logic as above, we see that the end-to-end delay is: You may now want to try to determine what the delay would be for P packets sent over a series of N links.

Queuing Delays and Packet Loss Each packet switch has multiple links attached to it. For each attached link, the packet switch has an output buffer (also called an output queue), which stores packets that the router is about to send into that link. The output buffers play a key role in packet switching. If an arriving packet needs to be transmitted onto a link but finds the link busy with the transmission of another packet, the arriving packet must wait in the output buffer. Thus, in addition to the store-and-forward delays, packets suffer output buffer queuing delays. These delays are variable and depend on the level of congestion in the network. end-to-end = NLR (1.1) Since the amount of buffer space is finite, an Figure 1.12 Packet switching arriving packet may find that the buffer is completely full with other packets waiting for transmission. In this case, packet loss will occur—either the arriving packet or one of the already-queued packets will be dropped. Figure 1.12 illustrates a simple packet-switched network. As in Figure 1.11, packets are represented by three-dimensional slabs. The width of a slab represents the number of bits in the packet. In this figure, all packets have the same width and hence the same length. Suppose Hosts A and B are sending packets to Host E. Hosts A and B first send their packets along 100 Mbps Ethernet links to the first router. The router then directs these packets to the 15 Mbps link. If, during a short interval of time, the arrival rate of packets to the router (when converted to bits per second) exceeds 15 Mbps, congestion will occur at the router as packets queue in the link's output buffer before being transmitted onto the link. For example, if Host A and B each send a burst of five packets back-to-back at the same time, then most of these packets will spend some time waiting in the queue. The situation is, in fact, entirely analogous to many common-day situations—for example, when we wait in line for a bank teller or wait in front of a tollbooth. We'll examine this queuing delay in more detail in Section 1.4.

Forwarding Tables and Routing Protocols Earlier, we said that a router takes a packet arriving on one of its attached communication links and forwards that packet onto another one of its attached communication links. But how does the router determine which link it should forward the packet onto? Packet forwarding is actually done in different ways in different types of computer networks. Here, we briefly describe how it

is done in the Internet. In the Internet, every end system has an address called an IP address. When a source end system wants to send a packet to a destination end system, the source includes the destination's IP address in the packet's header. As with postal addresses, this address has a hierarchical structure. When a packet arrives at a router in the network, the router examines a portion of the packet's destination address and forwards the packet to an adjacent router. More specifically, each router has a forwarding table that maps destination addresses (or portions of the destination addresses) to that router's outbound links. When a packet arrives at a router, the router examines the address and searches its forwarding table, using this destination address, to find the appropriate outbound link. The router then directs the packet to this outbound link. The end-to-end routing process is analogous to a car driver who does not use maps but instead prefers to ask for directions. For example, suppose Joe is driving from Philadelphia to 156 Lakeside Drive in Orlando, Florida. Joe first drives to his neighborhood gas station and asks how to get to 156 Lakeside Drive in Orlando, Florida. The gas station attendant extracts the Florida portion of the address and tells Joe that he needs to get onto the interstate highway I-95 South, which has an entrance just next to the gas station. He also tells Joe that once he enters Florida, he should ask someone else there. Joe then takes I-95 South until he gets to Jacksonville, Florida, at which point he asks another gas station attendant for directions. The attendant extracts the Orlando portion of the address and tells Joe that he should continue on I-95 to Daytona Beach and then ask someone else. In Daytona Beach, another gas station attendant also extracts the Orlando portion of the address and tells Joe that he should take I-4 directly to Orlando. Joe takes I-4 and gets off at the Orlando exit. Joe goes to another gas station attendant, and this time the attendant extracts the Lakeside Drive portion of the address and tells Joe the road he must follow to get to Lakeside Drive. Once Joe reaches Lakeside Drive, he asks a kid on a bicycle how to get to his destination. The kid extracts the 156 portion of the address and points to the house. Joe finally reaches his ultimate destination. In the above analogy, the gas station attendants and kids on bicycles are analogous to routers. We just learned that a router uses a packet's destination address to index a forwarding table and determine the appropriate outbound link. But this statement begs yet another question: How do forwarding tables get set? Are they configured by hand in each and every router, or does the Internet use a more automated procedure? This issue will be studied in depth in Chapter 5. But to whet your appetite here, we'll note now that the Internet has a number of special routing protocols that are used to automatically set the forwarding tables. A routing protocol may, for example, determine the shortest path from each router to each destination and use the shortest path results to configure the forwarding tables in the routers. How would you actually like to see the end-to-end route that packets take in the Internet? We now invite you to get your hands dirty by interacting with the Trace-route program. Simply visit the site www.traceroute.org, choose a source in a particular country, and trace the route from that source to your computer. (For a discussion of Traceroute, see Section 1.4.)

1.3.2 Circuit Switching

There are two fundamental approaches to moving data through a network of links and switches: circuit switching and packet switching. Having covered packet-switched networks in the previous subsection, we now turn our attention to circuit-switched networks. In circuit-switched networks, the resources needed along a path (buffers, link transmission rate) to provide for communication between the end systems are reserved for the duration of the communication session between the end systems. In packet-switched networks, these resources are not reserved; a session's messages use the resources on demand and, as a consequence, may have to wait (that is, queue) for access to a communication link. As a simple analogy, consider two restaurants, one that requires reservations and another that neither requires reservations nor accepts them. For the restaurant that requires reservations, we have

to go through the hassle of calling before we leave home. But when we arrive at the restaurant we can, in principle, immediately be seated and order our meal. For the restaurant that does not require reservations, we don't need to bother to reserve a table. But when we arrive at the restaurant, we may have to wait for a table before we can be seated. Traditional telephone networks are examples of circuit-switched networks. Consider what happens when one person wants to send information (voice or facsimile) to another over a telephone network. Before the sender can send the information, the network must establish a connection between the sender and the receiver. This is a bona fide connection for which the switches on the path between the sender and receiver maintain connection state for that connection. In the jargon of telephony, this connection is called a circuit. When the network establishes the circuit, it also reserves a constant transmission rate in the network's links (representing a fraction of each link's transmission capacity) for the duration of the connection. Since a given transmission rate has been reserved for this sender-to-receiver connection, the sender can transfer the data to the receiver at the guaranteed constant rate. Figure 1.13 illustrates a circuit-switched network. In this network, the four circuit switches are interconnected by four links. Each of these links has four circuits, so that each link can support four simultaneous connections. The hosts (for example, PCs and workstations) are each directly connected to one of the switches. When two hosts want to communicate, the network establishes a dedicated end-to-end connection between the two hosts. Thus, in order for Host A to communicate with Host B, the network must first reserve one circuit on each of two links. In this example, the dedicated end-to-end connection uses the second circuit in the first link and the fourth circuit in the second link. Because each link has four circuits, for each link used by the end-to-end connection, the connection gets one fourth of the link's total transmission capacity for the duration of the connection. Thus, for example, if each link between adjacent switches has a transmission rate of 1 Mbps, then each end-to-end circuit-switch connection gets 250 kbps of dedicated transmission rate. Figure 1.13 A simple circuit-switched network consisting of four switches and four links

In contrast, consider what happens when one host wants to send a packet to another host over a packet-switched network, such as the Internet. As with circuit switching, the packet is transmitted over a series of communication links. But different from circuit switching, the packet is sent into the network without reserving any link resources whatsoever. If one of the links is congested because other packets need to be transmitted over the link at the same time, then the packet will have to wait in a buffer at the sending side of the transmission link and suffer a delay. The Internet makes its best effort to deliver packets in a timely manner, but it does not make any guarantees.

Multiplexing in Circuit-Switched Networks

A circuit in a link is implemented with either frequency-division multiplexing (FDM) or time-division multiplexing (TDM). With FDM, the frequency spectrum of a link is divided up among the connections established across the link. Specifically, the link dedicates a frequency band to each connection for the duration of the connection. In telephone networks, this frequency band typically has a width of 4 kHz (that is, 4,000 hertz or 4,000 cycles per second). The width of the band is called, not surprisingly, the bandwidth. FM radio stations also use FDM to share the frequency spectrum between 88 MHz and 108 MHz, with each station being allocated a specific frequency band. For a TDM link, time is divided into frames of fixed duration, and each frame is divided into a fixed number of time slots. When the network establishes a connection across a link, the network dedicates one time slot in every frame to this connection. These slots are dedicated for the sole use of that connection, with one time slot available for use (in every frame) to transmit the connection's data. Figure 1.14

With FDM, each circuit continuously gets a fraction of the bandwidth. With TDM, each circuit gets all of the bandwidth periodically during brief intervals of time (that is, during slots) Figure 1.14 illustrates FDM and TDM for a specific

network link supporting up to four circuits. For FDM, the frequency domain is segmented into four bands, each of bandwidth 4 kHz. For TDM, the time domain is segmented into frames, with four time slots in each frame; each circuit is assigned the same dedicated slot in the revolving TDM frames. For TDM, the transmission rate of a circuit is equal to the frame rate multiplied by the number of bits in a slot. For example, if the link transmits 8,000 frames per second and each slot consists of 8 bits, then the transmission rate of each circuit is 64 kbps. Proponents of packet switching have always argued that circuit switching is wasteful because the dedicated circuits are idle during silent periods. For example, when one person in a telephone call stops talking, the idle network resources (frequency bands or time slots in the links along the connection's route) cannot be used by other ongoing connections. As another example of how these resources can be underutilized, consider a radiologist who uses a circuit-switched network to remotely access a series of x-rays. The radiologist sets up a connection, requests an image, contemplates the image, and then requests a new image. Network resources are allocated to the connection but are not used (i.e., are wasted) during the radiologist's contemplation periods. Proponents of packet switching also enjoy pointing out that establishing end-to-end circuits and reserving end-to-end transmission capacity is complicated and requires complex signaling software to coordinate the operation of the switches along the end-to-end path. Before we finish our discussion of circuit switching, let's work through a numerical example that should shed further insight on the topic. Let us consider how long it takes to send a file of 640,000 bits from Host A to Host B over a circuit-switched network. Suppose that all links in the network use TDM with 24 slots and have a bit rate of 1.536 Mbps. Also suppose that it takes 500 msec to establish an end-to-end circuit before Host A can begin to transmit the file. How long does it take to send the file? Each circuit has a transmission rate of so it takes seconds to transmit the file. To this 10 seconds we add the circuit establishment time, giving 10.5 seconds to send the file. Note that the transmission time is independent of the number of links: The transmission time would be 10 seconds if the end-to-end circuit passed through one link or a hundred links. (The actual $(1.536 \text{ Mbps})/24=64 \text{ kbps}$, $(640,000 \text{ bits})/(64 \text{ kbps})=10$ end-to-end delay also includes a propagation delay; see Section 1.4.)

Packet Switching Versus Circuit Switching Having described circuit switching and packet switching, let us compare the two. Critics of packet switching have often argued that packet switching is not suitable for real-time services (for example, telephone calls and video conference calls) because of its variable and unpredictable end-to-end delays (due primarily to variable and unpredictable queuing delays). Proponents of packet switching argue that (1) it offers better sharing of transmission capacity than circuit switching and (2) it is simpler, more efficient, and less costly to implement than circuit switching. An interesting discussion of packet switching versus circuit switching is [Molinero-Fernandez 2002]. Generally speaking, people who do not like to hassle with restaurant reservations prefer packet switching to circuit switching. Why is packet switching more efficient? Let's look at a simple example. Suppose users share a 1 Mbps link. Also suppose that each user alternates between periods of activity, when a user generates data at a constant rate of 100 kbps, and periods of inactivity, when a user generates no data. Suppose further that a user is active only 10 percent of the time (and is idly drinking coffee during the remaining 90 percent of the time). With circuit switching, 100 kbps must be reserved for each user at all times. For example, with circuit-switched TDM, if a one-second frame is divided into 10 time slots of 100 ms each, then each user would be allocated one time slot per frame. Thus, the circuit-switched link can support only simultaneous users. With packet switching, the probability that a specific user is active is 0.1 (that is, 10 percent). If there are 35 users, the probability that there are 11 or more simultaneously active users is approximately 0.0004. (Homework Problem P8 outlines how this

probability is obtained.) When there are 10 or fewer simultaneously active users (which happens with probability 0.9996), the aggregate arrival rate of data is less than or equal to 1 Mbps, the output rate of the link. Thus, when there are 10 or fewer active users, users' packets flow through the link essentially without delay, as is the case with circuit switching. When there are more than 10 simultaneously active users, then the aggregate arrival rate of packets exceeds the output capacity of the link, and the output queue will begin to grow. (It continues to grow until the aggregate input rate falls back below 1 Mbps, at which point the queue will begin to diminish in length.) Because the probability of having more than 10 simultaneously active users is minuscule in this example, packet switching provides essentially the same performance as circuit switching, but does so while allowing for more than three times the number of users. Let's now consider a second simple example. Suppose there are 10 users and that one user suddenly generates one thousand 1,000-bit packets, while other users remain quiescent and do not generate packets. Under TDM circuit switching with 10 slots per frame and each slot consisting of 1,000 bits, the active user can only use its one time slot per frame to transmit data, while the remaining nine time slots in each frame remain idle. It will be 10 seconds before all of the active user's one million bits of data has $10 (= 1 \text{ Mbps} / 100 \text{ kbps})$ been transmitted. In the case of packet switching, the active user can continuously send its packets at the full link rate of 1 Mbps, since there are no other users generating packets that need to be multiplexed with the active user's packets. In this case, all of the active user's data will be transmitted within 1 second. The above examples illustrate two ways in which the performance of packet switching can be superior to that of circuit switching. They also highlight the crucial difference between the two forms of sharing a link's transmission rate among multiple data streams. Circuit switching pre-allocates use of the transmission link regardless of demand, with allocated but unneeded link time going unused. Packet switching on the other hand allocates link use on demand. Link transmission capacity will be shared on a packet-by-packet basis only among those users who have packets that need to be transmitted over the link. Although packet switching and circuit switching are both prevalent in today's telecommunication networks, the trend has certainly been in the direction of packet switching. Even many of today's circuit-switched telephone networks are slowly migrating toward packet switching. In particular, telephone networks often use packet switching for the expensive overseas portion of a telephone call.

1.3.3 A Network of Networks

We saw earlier that end systems (PCs, smartphones, Web servers, mail servers, and so on) connect into the Internet via an access ISP. The access ISP can provide either wired or wireless connectivity, using an array of access technologies including DSL, cable, FTTH, Wi-Fi, and cellular. Note that the access ISP does not have to be a telco or a cable company; instead it can be, for example, a university (providing Internet access to students, staff, and faculty), or a company (providing access for its employees). But connecting end users and content providers into an access ISP is only a small piece of solving the puzzle of connecting the billions of end systems that make up the Internet. To complete this puzzle, the access ISPs themselves must be interconnected. This is done by creating a network of networks—understanding this phrase is the key to understanding the Internet. Over the years, the network of networks that forms the Internet has evolved into a very complex structure. Much of this evolution is driven by economics and national policy, rather than by performance considerations. In order to understand today's Internet network structure, let's incrementally build a series of network structures, with each new structure being a better approximation of the complex Internet that we have today. Recall that the overarching goal is to interconnect the access ISPs so that all end systems can send packets to each other. One naive approach would be to have each access ISP directly connect with every other access ISP. Such a mesh design is, of course, much too costly for the access ISPs, as it would require each

access ISP to have a separate communication link to each of the hundreds of thousands of other access ISPs all over the world. Our first network structure, Network Structure 1, interconnects all of the access ISPs with a single global transit ISP. Our (imaginary) global transit ISP is a network of routers and communication links that not only spans the globe, but also has at least one router near each of the hundreds of thousands of access ISPs. Of course, it would be very costly for the global ISP to build such an extensive network. To be profitable, it would naturally charge each of the access ISPs for connectivity, with the pricing reflecting (but not necessarily directly proportional to) the amount of traffic an access ISP exchanges with the global ISP. Since the access ISP pays the global transit ISP, the access ISP is said to be a customer and the global transit ISP is said to be a provider. Now if some company builds and operates a global transit ISP that is profitable, then it is natural for other companies to build their own global transit ISPs and compete with the original global transit ISP. This leads to Network Structure 2, which consists of the hundreds of thousands of access ISPs and multiple global transit ISPs. The access ISPs certainly prefer Network Structure 2 over Network Structure 1 since they can now choose among the competing global transit providers as a function of their pricing and services. Note, however, that the global transit ISPs themselves must interconnect: Otherwise access ISPs connected to one of the global transit providers would not be able to communicate with access ISPs connected to the other global transit providers. Network Structure 2, just described, is a two-tier hierarchy with global transit providers residing at the top tier and access ISPs at the bottom tier. This assumes that global transit ISPs are not only capable of getting close to each and every access ISP, but also find it economically desirable to do so. In reality, although some ISPs do have impressive global coverage and do directly connect with many access ISPs, no ISP has presence in each and every city in the world. Instead, in any given region, there may be a regional ISP to which the access ISPs in the region connect. Each regional ISP then connects to tier-1 ISPs. Tier-1 ISPs are similar to our (imaginary) global transit ISP; but tier-1 ISPs, which actually do exist, do not have a presence in every city in the world. There are approximately a dozen tier-1 ISPs, including Level 3 Communications, AT&T, Sprint, and NTT. Interestingly, no group officially sanctions tier-1 status; as the saying goes—if you have to ask if you're a member of a group, you're probably not. Returning to this network of networks, not only are there multiple competing tier-1 ISPs, there may be multiple competing regional ISPs in a region. In such a hierarchy, each access ISP pays the regional ISP to which it connects, and each regional ISP pays the tier-1 ISP to which it connects. (An access ISP can also connect directly to a tier-1 ISP, in which case it pays the tier-1 ISP). Thus, there is customer-provider relationship at each level of the hierarchy. Note that the tier-1 ISPs do not pay anyone as they are at the top of the hierarchy. To further complicate matters, in some regions, there may be a larger regional ISP (possibly spanning an entire country) to which the smaller regional ISPs in that region connect; the larger regional ISP then connects to a tier-1 ISP. For example, in China, there are access ISPs in each city, which connect to provincial ISPs, which in turn connect to national ISPs, which finally connect to tier-1 ISPs [Tian 2012]. We refer to this multi-tier hierarchy, which is still only a crude approximation of today's Internet, as Network Structure 3. To build a network that more closely resembles today's Internet, we must add points of presence (PoPs), multi-homing, peering, and Internet exchange points (IXPs) to the hierarchical Network Structure 3. PoPs exist in all levels of the hierarchy, except for the bottom (access ISP) level. A PoP is simply a group of one or more routers (at the same location) in the provider's network where customer ISPs can connect into the provider ISP. For a customer network to connect to a provider's PoP, it can lease a high-speed link from a third-party telecommunications provider to directly connect one of its routers to a router at the PoP. Any ISP (except for tier-1 ISPs) may choose to multi-home, that is, to

connect to two or more provider ISPs. So, for example, an access ISP may multi-home with two regional ISPs, or it may multi-home with two regional ISPs and also with a tier-1 ISP. Similarly, a regional ISP may multi-home with multiple tier-1 ISPs. When an ISP multi-homes, it can continue to send and receive packets into the Internet even if one of its providers has a failure. As we just learned, customer ISPs pay their provider ISPs to obtain global Internet interconnectivity. The amount that a customer ISP pays a provider ISP reflects the amount of traffic it exchanges with the provider. To reduce these costs, a pair of nearby ISPs at the same level of the hierarchy can peer, that is, they can directly connect their networks together so that all the traffic between them passes over the direct connection rather than through upstream intermediaries. When two ISPs peer, it is typically settlement-free, that is, neither ISP pays the other. As noted earlier, tier-1 ISPs also peer with one another, settlement-free. For a readable discussion of peering and customer-provider relationships, see [Van der Berg 2008]. Along these same lines, a third-party company can create an Internet Exchange Point (IXP), which is a meeting point where multiple ISPs can peer together. An IXP is typically in a stand-alone building with its own switches [Ager 2012]. There are over 400 IXPs in the Internet today [IXP List 2016]. We refer to this ecosystem—consisting of access ISPs, regional ISPs, tier-1 ISPs, PoPs, multi-homing, peering, and IXPs—as Network Structure 4. We now finally arrive at Network Structure 5, which describes today’s Internet. Network Structure 5, illustrated in Figure 1.15, builds on top of Network Structure 4 by adding content-provider networks. Google is currently one of the leading examples of such a content-provider network. As of this writing, it is estimated that Google has 50–100 data centers distributed across North America, Europe, Asia, South America, and Australia. Some of these data centers house over one hundred thousand servers, while other data centers are smaller, housing only hundreds of servers. The Google data centers are all interconnected via Google’s private TCP/IP network, which spans the entire globe but is nevertheless separate from the public Internet. Importantly, the Google private network only carries traffic to/from Google servers. As shown in Figure 1.15, the Google private network attempts to “bypass” the upper tiers of the Internet by peering (settlement free) with lower-tier ISPs, either by directly connecting with them or by connecting with them at IXPs [Labovitz 2010]. However, because many access ISPs can still only be reached by transiting through tier-1 networks, the Google network also connects to tier-1 ISPs, and pays those ISPs for the traffic it exchanges with them. By creating its own network, a content provider not only reduces its payments to upper-tier ISPs, but also has greater control of how its services are ultimately delivered to end users. Google’s network infrastructure is described in greater detail in Section 2.6. In summary, today’s Internet—a network of networks—is complex, consisting of a dozen or so tier-1 ISPs and hundreds of thousands of lower-tier ISPs. The ISPs are diverse in their coverage, with some spanning multiple continents and oceans, and others limited to narrow geographic regions. The lower-tier ISPs connect to the higher-tier ISPs, and the higher-tier ISPs interconnect with one another. Users and content providers are customers of lower-tier ISPs, and lower-tier ISPs are customers of higher-tier ISPs. In recent years, major content providers have also created their own networks and connect directly into lower-tier ISPs where possible.

Figure 1.15 Interconnection of ISPs

1.4 Delay, Loss, and Throughput in Packet-Switched Networks

Back in Section 1.1 we said that the Internet can be viewed as an infrastructure that provides services to distributed applications running on end systems. Ideally, we would like Internet services to be able to move as much data as we want between any two end systems, instantaneously, without any loss of data. Alas, this is a lofty goal, one that is unachievable in reality. Instead, computer networks necessarily constrain throughput (the amount of data per second that can be transferred) between end systems, introduce delays between end systems, and can actually lose packets. On one hand, it is unfortunate that

the physical laws of reality introduce delay and loss as well as constrain throughput. On the other hand, because computer networks have these problems, there are many fascinating issues surrounding how to deal with the problems—more than enough issues to fill a course on computer networking and to motivate thousands of PhD theses! In this section, we'll begin to examine and quantify delay, loss, and throughput in computer networks.

1.4.1 Overview of Delay in Packet-Switched Networks

Recall that a packet starts in a host (the source), passes through a series of routers, and ends its journey in another host (the destination). As a packet travels from one node (host or router) to the subsequent node (host or router) along this path, the packet suffers from several types of delays at each node along the path. The most important of these delays are the nodal processing delay, queuing delay, transmission delay, and propagation delay; together, these delays accumulate to give a total nodal delay. The performance of many Internet applications—such as search, Web browsing, e-mail, maps, instant messaging, and voice-over-IP—are greatly affected by network delays. In order to acquire a deep understanding of packet switching and computer networks, we must understand the nature and importance of these delays.

Types of Delay

Let's explore these delays in the context of Figure 1.16. As part of its end-to-end route between source and destination, a packet is sent from the upstream node through router A to router B. Our goal is to characterize the nodal delay at router A. Note that router A has an outbound link leading to router B. This link is preceded by a queue (also known as a buffer). When the packet arrives at router A from the upstream node, router A examines the packet's header to determine the appropriate outbound link for the packet and then directs the packet to this link. In this example, the outbound link for the packet is the one that leads to router B. A packet can be transmitted on a link only if there is no other packet currently being transmitted on the link and if there are no other packets preceding it in the queue; if the link is Figure 1.16 The nodal delay at router A currently busy or if there are other packets already queued for the link, the newly arriving packet will then join the queue.

Processing Delay

The time required to examine the packet's header and determine where to direct the packet is part of the processing delay. The processing delay can also include other factors, such as the time needed to check for bit-level errors in the packet that occurred in transmitting the packet's bits from the upstream node to router A. Processing delays in high-speed routers are typically on the order of microseconds or less. After this nodal processing, the router directs the packet to the queue that precedes the link to router B. (In Chapter 4 we'll study the details of how a router operates.)

Queuing Delay

At the queue, the packet experiences a queuing delay as it waits to be transmitted onto the link. The length of the queuing delay of a specific packet will depend on the number of earlier-arriving packets that are queued and waiting for transmission onto the link. If the queue is empty and no other packet is currently being transmitted, then our packet's queuing delay will be zero. On the other hand, if the traffic is heavy and many other packets are also waiting to be transmitted, the queuing delay will be long. We will see shortly that the number of packets that an arriving packet might expect to find is a function of the intensity and nature of the traffic arriving at the queue. Queuing delays can be on the order of microseconds to milliseconds in practice.

Transmission Delay

Assuming that packets are transmitted in a first-come-first-served manner, as is common in packet-switched networks, our packet can be transmitted only after all the packets that have arrived before it have been transmitted. Denote the length of the packet by L bits, and denote the transmission rate of the link from router A to router B by R bits/sec. For example, for a 10 Mbps Ethernet link, the rate is for a 100 Mbps Ethernet link, the rate is The transmission delay is L/R . This is the amount of time required to push (that is, transmit) all of the packet's bits into the link. Transmission delays are typically on the order of microseconds to milliseconds in practice.

Propagation Delay

Once a bit is pushed into the link,

it needs to propagate to router B. The time required to propagate from the beginning of the link to router B is the propagation delay. The bit propagates at the propagation speed of the link. The propagation speed depends on the physical medium of the link (that is, fiber optics, twisted-pair copper wire, and so on) and is in the range of which is equal to, or a little less than, the speed of light. The propagation delay is the distance between two routers divided by the propagation speed. That is, the propagation delay is d/s , where d is the distance between router A and router B and s is the propagation speed of the link. Once the last bit of the packet propagates to node B, it and all the preceding bits of the packet are stored in router B. The whole process then continues with router B now performing the forwarding. In wide-area networks, propagation delays are on the order of milliseconds.

Comparing Transmission and Propagation Delay Exploring propagation delay and transmission delay Newcomers to the field of computer networking sometimes have difficulty understanding the difference between transmission delay and propagation delay. The difference is subtle but important. The transmission delay is the amount of time required for the router to push out the packet; it is a function of the packet's length and the transmission rate of the link, but has nothing to do with the distance between the two routers. The propagation delay, on the other hand, is the time it takes a bit to propagate from one router to the next; it is a function of the distance between the two routers, but has nothing to do with the packet's length or the transmission rate of the link.

An analogy might clarify the notions of transmission and propagation delay. Consider a highway that has a tollbooth every 100 kilometers, as shown in Figure 1.17. You can think of the highway segments as links and the tollbooths as routers. Suppose that cars travel (that is, propagate) on the highway at a rate of 100 km/hour (that is, when a car leaves a tollbooth, it instantaneously accelerates to 100 km/hour and maintains that speed between tollbooths). Suppose next that 10 cars, traveling together as a caravan, follow each other in a fixed order. You can think of each car as a bit and the caravan as a packet. Also suppose that each tollbooth services (that is, transmits) a car at a rate of one car per 12 seconds, and that it is late at night so that the caravan's cars are the only cars on the highway. Finally, suppose that whenever the first car of the caravan arrives at a tollbooth, it waits at the entrance until the other nine cars have arrived and lined up behind it. (Thus the entire caravan must be stored at the tollbooth before it can begin to be forwarded.) The time required for the tollbooth to push the entire caravan onto the highway is . This time is analogous to the transmission delay in a router. The time required for a car to travel from the exit of one tollbooth to the next tollbooth is . This time is analogous to propagation delay. Therefore, the time from when the caravan is stored in front of a tollbooth until the caravan is stored in front of the next tollbooth is the sum of transmission delay and propagation delay—in this example, 62 minutes. Let's explore this analogy a bit more. What would happen if the tollbooth service time for a caravan were greater than the time for a car to travel between tollbooths? For example, suppose now that the cars travel at the rate of 1,000 km/hour and the tollbooth services cars at the rate of one car per minute. Then the traveling delay between two tollbooths is 6 minutes and the time to serve a caravan is 10 minutes. In this case, the first few cars in the caravan will arrive at the second tollbooth before the last cars in the caravan leave the first tollbooth. This situation also arises in packet-switched networks—the first bits in a packet can arrive at a router while many of the remaining bits in the packet are still waiting to be transmitted by the preceding router. If a picture speaks a thousand words, then an animation must speak a million words. The Web site for this textbook provides an interactive Java applet that nicely illustrates and contrasts transmission delay and propagation delay. The reader is highly encouraged to visit that applet. [Smith 2009] also provides a very readable discussion of propagation, queueing, and

transmission delays. If we let d_p , d_q , d_t , and d_{prop} denote the processing, queuing, transmission, and propagation delays, then the total nodal delay is given by $d_{nodal} = d_p + d_q + d_t + d_{prop}$. The contribution of these delay components can vary significantly. For example, d_t can be negligible (for example, a couple of microseconds) for a link connecting two routers on the same university campus; however, d_t is hundreds of milliseconds for two routers interconnected by a geostationary satellite link, and d_{prop} can be the dominant term in d_{nodal} . Similarly, d_q can range from negligible to significant. Its contribution is typically negligible for transmission rates of 10 Mbps and higher (for example, for LANs); however, it can be hundreds of milliseconds for large Internet packets sent over low-speed dial-up modem links. The processing delay, d_p , is often negligible; however, it strongly influences a router's maximum throughput, which is the maximum rate at which a router can forward packets.

1.4.2 Queuing Delay and Packet Loss

The most complicated and interesting component of nodal delay is the queuing delay, d_q . In fact, queuing delay is so important and interesting in computer networking that thousands of papers and numerous books have been written about it [Bertsekas 1991; Daigle 1991; Kleinrock 1975, Kleinrock 1976; Ross 1995]. We give only a high-level, intuitive discussion of queuing delay here; the more curious reader may want to browse through some of the books (or even eventually write a PhD thesis on the subject!). Unlike the other three delays (namely, d_p , d_t , and d_{prop}), the queuing delay can vary from packet to packet. For example, if 10 packets arrive at an empty queue at the same time, the first packet transmitted will suffer no queuing delay, while the last packet transmitted will suffer a relatively large queuing delay (while it waits for the other nine packets to be transmitted). Therefore, when characterizing queuing delay, one typically uses statistical measures, such as average queuing delay, variance of queuing delay, and the probability that the queuing delay exceeds some specified value. When is the queuing delay large and when is it insignificant? The answer to this question depends on the rate at which traffic arrives at the queue, the transmission rate of the link, and the nature of the arriving traffic, that is, whether the traffic arrives periodically or arrives in bursts. To gain some insight here, let λ denote the average rate at which packets arrive at the queue (λ is in units of packets/sec). Recall that R is the transmission rate; that is, it is the rate (in bits/sec) at which bits are pushed out of the queue. Also suppose, for simplicity, that all packets consist of L bits. Then the average rate at which bits arrive at the queue is λL bits/sec. Finally, assume that the queue is very big, so that it can hold essentially an infinite number of bits. The ratio $\lambda L / R$, called the traffic intensity, often plays an important role in estimating the extent of the queuing delay. If $\lambda L / R > 1$, then the average rate at which bits arrive at the queue exceeds the rate at which the bits can be transmitted from the queue. In this unfortunate situation, the queue will tend to increase without bound and the queuing delay will approach infinity! Therefore, one of the golden rules in traffic engineering is: Design your system so that the traffic intensity is no greater than 1. Now consider the case $\lambda L / R \leq 1$. Here, the nature of the arriving traffic impacts the queuing delay. For example, if packets arrive periodically—that is, one packet arrives every L / R seconds—then every packet will arrive at an empty queue and there will be no queuing delay. On the other hand, if packets arrive in bursts but periodically, there can be a significant average queuing delay. For example, suppose N packets arrive simultaneously every $(L / R)N$ seconds. Then the first packet transmitted has no queuing delay; the second packet transmitted has a queuing delay of L / R seconds; and more generally, the n th packet transmitted has a queuing delay of $(n-1)L / R$ seconds. We leave it as an exercise for you to calculate the average queuing delay in this example. The two examples of periodic arrivals described above are a bit academic. Typically, the arrival process to a queue is random; that is, the arrivals do not follow any pattern and the packets are spaced apart by

random amounts of time. In this more realistic case, the quantity $\lambda a/R$ is not usually sufficient to fully characterize the queuing delay statistics. Nonetheless, it is useful in gaining an intuitive understanding of the extent of the queuing delay. In particular, if the traffic intensity is close to zero, then packet arrivals are few and far between and it is unlikely that an arriving packet will find another packet in the queue. Hence, the average queuing delay will be close to zero. On the other hand, when the traffic intensity is close to 1, there will be intervals of time when the arrival rate exceeds the transmission capacity (due to variations in packet arrival rate), and a queue will form during these periods of time; when the arrival rate is less than the transmission capacity, the length of the queue will shrink. Nonetheless, as the traffic intensity approaches 1, the average queue length gets larger and larger. The qualitative dependence of average queuing delay on the traffic intensity is shown in Figure 1.18. One important aspect of Figure 1.18 is the fact that as the traffic intensity approaches 1, the average queuing delay increases rapidly. A small percentage increase in the intensity will result in a much larger percentage-wise increase in delay. Perhaps you have experienced this phenomenon on the highway. If you regularly drive on a road that is typically congested, the fact that the road is typically (n-1) Figure 1.18

Dependence of average queuing delay on traffic intensity congested means that its traffic intensity is close to 1. If some event causes an even slightly larger-than-usual amount of traffic, the delays you experience can be huge. To really get a good feel for what queuing delays are about, you are encouraged once again to visit the textbook Web site, which provides an interactive Java applet for a queue. If you set the packet arrival rate high enough so that the traffic intensity exceeds 1, you will see the queue slowly build up over time. Packet Loss In our discussions above, we have assumed that the queue is capable of holding an infinite number of packets. In reality a queue preceding a link has finite capacity, although the queuing capacity greatly depends on the router design and cost. Because the queue capacity is finite, packet delays do not really approach infinity as the traffic intensity approaches 1. Instead, a packet can arrive to find a full queue. With no place to store such a packet, a router will drop that packet; that is, the packet will be lost. This overflow at a queue can again be seen in the Java applet for a queue when the traffic intensity is greater than 1. From an end-system viewpoint, a packet loss will look like a packet having been transmitted into the network core but never emerging from the network at the destination. The fraction of lost packets increases as the traffic intensity increases. Therefore, performance at a node is often measured not only in terms of delay, but also in terms of the probability of packet loss. As we'll discuss in the subsequent chapters, a lost packet may be retransmitted on an end-to-end basis in order to ensure that all data are eventually transferred from source to destination.

1.4.3 End-to-End Delay

Our discussion up to this point has focused on the nodal delay, that is, the delay at a single router. Let's now consider the total delay from source to destination. To get a handle on this concept, suppose there are routers between the source host and the destination host. Let's also suppose for the moment that the network is uncongested (so that queuing delays are negligible), the processing delay at each router and at the source host is d , the transmission rate out of each router and out of the source host is R bits/sec, and the propagation on each link is d . The nodal delays accumulate and give an end-to-end delay, where, once again, where L is the packet size. Note that Equation 1.2 is a generalization of Equation 1.1, which did not take into account processing and propagation delays. We leave it to you to generalize Equation 1.2 to the case of heterogeneous delays at the nodes and to the presence of an average queuing delay at each node.

Traceroute

Using Traceroute to discover network paths and measure network delay To get a hands-on feel for end-to-end delay in a computer network, we can make use of the Traceroute program. Traceroute is a simple program that can run in any Internet host. When the user specifies a destination hostname, the program in the source host sends

multiple, special packets toward that destination. As these packets work their way toward the destination, they pass through a series of routers. When a router receives one of these special packets, it sends back to the source a short message that contains the name and address of the router. More specifically, suppose there are routers between the source and the destination. Then the source will send N special packets into the network, with each packet addressed to the ultimate destination. These N special packets are marked 1 through N , with the first packet marked 1 and the last packet marked N . When the n th router receives the n th packet marked n , the router does not forward the packet toward its destination, but instead sends a message back to the source. When the destination host receives the N th packet, it too returns a message back to the source. The source records the time that elapses between when it sends a packet and when it receives the corresponding $N-1$ packet.

$d_{end-end} = N(d_{proc} + d_{trans} + d_{prop})$ (1.2) $d_{trans} = L/R$, $N-1$ return message; it also records the name and address of the router (or the destination host) that returns the message. In this manner, the source can reconstruct the route taken by packets flowing from source to destination, and the source can determine the round-trip delays to all the intervening routers. Traceroute actually repeats the experiment just described three times, so the source actually sends $3 \cdot N$ packets to the destination. RFC 1393 describes Traceroute in detail. Here is an example of the output of the Traceroute program, where the route was being traced from the source host `gaia.cs.umass.edu` (at the University of Massachusetts) to the host `cis.poly.edu` (at Polytechnic University in Brooklyn). The output has six columns: the first column is the n value described above, that is, the number of the router along the route; the second column is the name of the router; the third column is the address of the router (of the form `xxx.xxx.xxx.xxx`); the last three columns are the round-trip delays for three experiments. If the source receives fewer than three messages from any given router (due to packet loss in the network), Traceroute places an asterisk just after the router number and reports fewer than three round-trip times for that router.

```

1 cs-gw (128.119.240.254) 1.009 ms 0.899 ms 0.993 ms
2 128.119.3.154 (128.119.3.154) 0.931 ms 0.441 ms 0.651 ms
3 -border4-rt-gi-1-3.gw.umass.edu (128.119.2.194) 1.032 ms 0.484 ms 0.451 ms
4 -acr1-ge-2-1-0.Boston.cw.net (208.172.51.129) 10.006 ms 8.150 ms 8.460 ms
5 -agr4-loopback.NewYork.cw.net (206.24.194.104) 12.272 ms 14.344 ms 13.267 ms
6 -acr2-loopback.NewYork.cw.net (206.24.194.62) 13.225 ms 12.292 ms 12.148 ms
7 -pos10-2.core2.NewYork1.Level3.net (209.244.160.133) 12.218 ms 11.823 ms 11.793 ms
8 -gige9-1-52.hsipaccess1.NewYork1.Level3.net (64.159.17.39) 13.081 ms 11.556 ms 13.297 ms
9 -p0-0.polyu.bbnplanet.net (4.25.109.122) 12.716 ms 13.052 ms 12.786 ms
10 cis.poly.edu (128.238.32.126) 14.080 ms 13.035 ms 12.802 ms
  
```

In the trace above there are nine routers between the source and the destination. Most of these routers have a name, and all of them have addresses. For example, the name of Router 3 is `border4-rt-gi-1-3.gw.umass.edu` and its address is `128.119.2.194`. Looking at the data provided for this same router, we see that in the first of the three trials the round-trip delay between the source and the router was 1.03 msec. The round-trip delays for the subsequent two trials were 0.48 and 0.45 msec. These round-trip delays include all of the delays just discussed, including transmission delays, propagation delays, router processing delays, and queuing delays. Because the queuing delay is varying with time, the round-trip delay of packet n sent to a router n can sometimes be longer than the round-trip delay of packet $n+1$ sent to router $n+1$. Indeed, we observe this phenomenon in the above example: the delays to Router 6 are larger than the delays to Router 7!

Want to try out Traceroute for yourself? We highly recommended that you visit <http://www.traceroute.org>, which provides a Web interface to an extensive list of sources for route tracing. You choose a source and supply the hostname for any destination. The Traceroute program then does all the work. There are a number of free software programs that provide a

graphical interface to Traceroute; one of our favorites is PingPlotter [PingPlotter 2016].

End System, Application, and Other Delays

In addition to processing, transmission, and propagation delays, there can be additional significant delays in the end systems. For example, an end system wanting to transmit a packet into a shared medium (e.g., as in a WiFi or cable modem scenario) may purposefully delay its transmission as part of its protocol for sharing the medium with other end systems; we'll consider such protocols in detail in Chapter 6. Another important delay is media packetization delay, which is present in Voice-over-IP (VoIP) applications. In VoIP, the sending side must first fill a packet with encoded digitized speech before passing the packet to the Internet. This time to fill a packet—called the packetization delay—can be significant and can impact the user-perceived quality of a VoIP call. This issue will be further explored in a homework problem at the end of this chapter.

1.4.4 Throughput in Computer Networks

In addition to delay and packet loss, another critical performance measure in computer networks is end-to-end throughput. To define throughput, consider transferring a large file from Host A to Host B across a computer network. This transfer might be, for example, a large video clip from one peer to another in a P2P file sharing system. The instantaneous throughput at any instant of time is the rate (in bits/sec) at which Host B is receiving the file. (Many applications, including many P2P file sharing systems, display the instantaneous throughput during downloads in the user interface—perhaps you have observed this before!) If the file consists of F bits and the transfer takes T seconds for Host B to receive all F bits, then the average throughput of the file transfer is F/T bits/sec. For some applications, such as Internet telephony, it is desirable to have a low delay and an instantaneous throughput consistently above some threshold (for example, over 24 kbps for some Internet telephony applications and over 256 kbps for some real-time video applications). For other applications, including those involving file transfers, delay is not critical, but it is desirable to have the highest possible throughput. To gain further insight into the important concept of throughput, let's consider a few examples.

Figure 1.19(a) shows two end systems, a server and a client, connected by two communication links and a router. Consider the throughput for a file transfer from the server to the client. Let R_1 denote the rate of the link between the server and the router; and R_2 denote the rate of the link between the router and the client. Suppose that the only bits being sent in the entire network are those from the server to the client. We now ask, in this ideal scenario, what is the server-to-client throughput? To answer this question, we may think of bits as fluid and communication links as pipes. Clearly, the server cannot pump bits through its link at a rate faster than R_1 bps; and the router cannot forward bits at a rate faster than R_2 bps. If then the bits pumped by the server will “flow” right through the router and arrive at the client at a rate of R_2 bps, giving a throughput of R_2 bps. If, on the other hand, then the router will not be able to forward bits as quickly as it receives them. In this case, bits will only leave the router at rate R_2 , giving an end-to-end throughput of R_2 . (Note also that if bits continue to arrive at the router at rate R_1 , and continue to leave the router at R_2 , the backlog of bits at the router waiting

Figure 1.19 Throughput for a file transfer from server to client for transmission to the client will grow and grow—a most undesirable situation!) Thus, for this simple two-link network, the throughput is $\min\{R_1, R_2\}$, that is, it is the transmission rate of the bottleneck link. Having determined the throughput, we can now approximate the time it takes to transfer a large file of F bits from server to client as $F/\min\{R_1, R_2\}$. For a specific example, suppose you are downloading an MP3 file of million bits, the server has a transmission rate of Mbps, and you have an access link of Mbps. The time needed to transfer the file is then 32 seconds. Of course, these expressions for throughput and transfer time are only approximations, as they do not account for store-and-forward and processing delays as well as protocol issues. Figure 1.19(b) now shows a network with N links between the server and the client, with the transmission rates of the N links being

Applying the same analysis as for the two-link network, we find that the throughput for a file transfer from server to client is $\min(s, c, R)$, which is $\min(250, 354, 250) = 250$ bytes/sec. The example SMTP session is as follows:

```

S: 250 alice@crepes.fr ... Sender ok
C: RCPT TO: S: 250 bob@hamburger.edu ... Recipient ok
C: DATA S: 354 Enter mail, end with "." on
a line by itself
C: Do you like ketchup? C: How about pickles? C: .
S: 250 Message accepted for
delivery
C: QUIT S: 221 hamburger.edu closing connection

```

In the example above, the client sends a message ("Do you like ketchup? How about pickles?") from mail server crepes.fr to mail server hamburger.edu. As part of the dialogue, the client issued five commands: HELO (an abbreviation for HELLO), MAIL FROM, RCPT TO, DATA, and QUIT. These commands are self-explanatory. The client also sends a line consisting of a single period, which indicates the end of the message to the server. (In ASCII jargon, each message ends with CRLF.CRLF, where CR and LF stand for carriage return and line feed, respectively.) The server issues replies to each command, with each reply having a reply code and some (optional) English language explanation. We mention here that SMTP uses persistent connections: If the sending mail server has several messages to send to the same receiving mail server, it can send all of the messages over the same TCP connection. For each message, the client begins the process with a new MAIL FROM: crepes.fr, designates the end of message with an isolated period, and issues QUIT only after all messages have been sent. It is highly recommended that you use Telnet to carry out a direct dialogue with an SMTP server. To do this, issue telnet serverName 25 where serverName is the name of a local mail server. When you do this, you are simply establishing a TCP connection between your local host and the mail server. After typing this line, you should immediately receive the 220 reply from the server. Then issue the SMTP commands HELO, MAIL FROM, RCPT TO, DATA, CRLF.CRLF, and QUIT at the appropriate times. It is also highly recommended that you do Programming Assignment 3 at the end of this chapter. In that assignment, you'll build a simple user agent that implements the client side of SMTP. It will allow you to send an e-mail message to an arbitrary recipient via a local mail server.

2.3.2 Comparison with HTTP

Let's now briefly compare SMTP with HTTP. Both protocols are used to transfer files from one host to another: HTTP transfers files (also called objects) from a Web server to a Web client (typically a browser); SMTP transfers files (that is, e-mail messages) from one mail server to another mail server. When transferring the files, both persistent HTTP and SMTP use persistent connections. Thus, the two protocols have common characteristics. However, there are important differences. First, HTTP is mainly a pull protocol—someone loads information on a Web server and users use HTTP to pull the information from the server at their convenience. In particular, the TCP connection is initiated by the machine that wants to receive the file. On the other hand, SMTP is primarily a push protocol—the sending mail server pushes the file to the receiving mail server. In particular, the TCP connection is initiated by the machine that wants to send the file. A second difference, which we alluded to earlier, is that SMTP requires each message, including the body of each message, to be in 7-bit ASCII format. If the message contains characters that are not 7-bit ASCII (for example, French characters with accents) or contains binary data (such as an image file), then the message has to be encoded into 7-bit ASCII. HTTP data does not impose this restriction. A third important difference concerns how a document consisting of text and images (along with possibly other media types) is handled. As we learned in Section 2.2, HTTP encapsulates each object in its own HTTP response message. SMTP places all of the message's objects into one message.

2.3.3 Mail Message Formats

When Alice writes an ordinary snail-mail letter to Bob, she may include all kinds of peripheral header information at the top of the letter, such as Bob's address, her own return address, and the date. Similarly, when an e-mail message is sent from one person to another, a header containing peripheral information precedes the body of the message itself. This peripheral information is contained in a series of

header lines, which are defined in RFC 5322. The header lines and the body of the message are separated by a blank line (that is, by CRLF). RFC 5322 specifies the exact format for mail header lines as well as their semantic interpretations. As with HTTP, each header line contains readable text, consisting of a keyword followed by a colon followed by a value. Some of the keywords are required and others are optional. Every header must have a From: header line and a To: header line; a header may include a Subject: header line as well as other optional header lines. It is important to note that these header lines are different from the SMTP commands we studied in Section 2.4.1 (even though they contain some common words such as “from” and “to”). The commands in that section were part of the SMTP handshaking protocol; the header lines examined in this section are part of the mail message itself. A typical message header looks like this: From: alice@crepes.fr To: bob@hamburger.edu Subject: Searching for the meaning of life. After the message header, a blank line follows; then the message body (in ASCII) follows. You should use Telnet to send a message to a mail server that contains some header lines, including the Subject: header line. To do this, issue telnet serverName 25, as discussed in Section 2.4.1.

2.3.4 Mail Access Protocols

Once SMTP delivers the message from Alice’s mail server to Bob’s mail server, the message is placed in Bob’s mailbox. Throughout this discussion we have tacitly assumed that Bob reads his mail by logging onto the server host and then executing a mail reader that runs on that host. Up until the early 1990s this was the standard way of doing things. But today, mail access uses a client-server architecture—the typical user reads e-mail with a client that executes on the user’s end system, for example, on an office PC, a laptop, or a smartphone. By executing a mail client on a local PC, users enjoy a rich set of features, including the ability to view multimedia messages and attachments. Given that Bob (the recipient) executes his user agent on his local PC, it is natural to consider placing a mail server on his local PC as well. With this approach, Alice’s mail server would dialogue directly with Bob’s PC. There is a problem with this approach, however. Recall that a mail server manages mailboxes and runs the client and server sides of SMTP. If Bob’s mail server were to reside on his local PC, then Bob’s PC would have to remain always on, and connected to the Internet, in order to receive new mail, which can arrive at any time. This is impractical for many Internet users. Instead, a typical user runs a user agent on the local PC but accesses its mailbox stored on an always-on shared mail server. This mail server is shared with other users and is typically maintained by the user’s ISP (for example, university or company). Now let’s consider the path an e-mail message takes when it is sent from Alice to Bob. We just learned that at some point along the path the e-mail message needs to be deposited in Bob’s mail server. This could be done simply by having Alice’s user agent send the message directly to Bob’s mail server. And this could be done with SMTP—indeed, SMTP has been designed for pushing e-mail from one host to another. However, typically the sender’s user agent does not dialogue directly with the recipient’s mail server. Instead, as shown in Figure 2.16, Alice’s user agent uses SMTP to push the e-mail message into her mail server, then Alice’s mail server uses SMTP (as an SMTP client) to relay the e-mail message to Bob’s mail server. Why the two-step procedure? Primarily because without relaying through Alice’s mail server, Alice’s user agent doesn’t have any recourse to an unreachable destination.

Figure 2.16 E-mail protocols and their communicating entities

mail server. By having Alice first deposit the e-mail in her own mail server, Alice’s mail server can repeatedly try to send the message to Bob’s mail server, say every 30 minutes, until Bob’s mail server becomes operational. (And if Alice’s mail server is down, then she has the recourse of complaining to her system administrator!) The SMTP RFC defines how the SMTP commands can be used to relay a message across multiple SMTP servers. But there is still one missing piece to the puzzle! How does a recipient like Bob, running a user agent on his local PC, obtain his messages, which are sitting in a mail server within Bob’s

ISP? Note that Bob's user agent can't use SMTP to obtain the messages because obtaining the messages is a pull operation, whereas SMTP is a push protocol. The puzzle is completed by introducing a special mail access protocol that transfers messages from Bob's mail server to his local PC. There are currently a number of popular mail access protocols, including Post Office Protocol—Version 3 (POP3), Internet Mail Access Protocol (IMAP), and HTTP. Figure 2.16 provides a summary of the protocols that are used for Internet mail: SMTP is used to transfer mail from the sender's mail server to the recipient's mail server; SMTP is also used to transfer mail from the sender's user agent to the sender's mail server. A mail access protocol, such as POP3, is used to transfer mail from the recipient's mail server to the recipient's user agent.

POP3 POP3 is an extremely simple mail access protocol. It is defined in [RFC 1939], which is short and quite readable. Because the protocol is so simple, its functionality is rather limited. POP3 begins when the user agent (the client) opens a TCP connection to the mail server (the server) on port 110. With the TCP connection established, POP3 progresses through three phases: authorization, transaction, and update. During the first phase, authorization, the user agent sends a username and a password (in the clear) to authenticate the user. During the second phase, transaction, the user agent retrieves messages; also during this phase, the user agent can mark messages for deletion, remove deletion marks, and obtain mail statistics. The third phase, update, occurs after the client has issued the quit command, ending the POP3 session; at this time, the mail server deletes the messages that were marked for deletion. In a POP3 transaction, the user agent issues commands, and the server responds to each command with a reply. There are two possible responses: +OK (sometimes followed by server-to-client data), used by the server to indicate that the previous command was fine; and -ERR, used by the server to indicate that something was wrong with the previous command. The authorization phase has two principal commands: user and pass. To illustrate these two commands, we suggest that you Telnet directly into a POP3 server, using port 110, and issue these commands. Suppose that mailServer is the name of your mail server. You will see something like: telnet mailServer 110 +OK POP3 server ready user bob +OK pass hungry +OK user successfully logged on If you misspell a command, the POP3 server will reply with an -ERR message. Now let's take a look at the transaction phase. A user agent using POP3 can often be configured (by the user) to "download and delete" or to "download and keep." The sequence of commands issued by a POP3 user agent depends on which of these two modes the user agent is operating in. In the download-and-delete mode, the user agent will issue the list, retr, and dele commands. As an example, suppose the user has two messages in his or her mailbox. In the dialogue below, C: (standing for client) is the user agent and S: (standing for server) is the mail server. The transaction will look something like: C: list S: 1 498 S: 2 912 S: . C: retr 1 S: (blah blah ... S: S:blah) S: . C: dele 1 C: retr 2 S: (blah blah ... S: S:blah) S: . C: dele 2 C: quit S: +OK POP3 server signing off The user agent first asks the mail server to list the size of each of the stored messages. The user agent then retrieves and deletes each message from the server. Note that after the authorization phase, the user agent employed only four commands: list, retr, dele, and quit. The syntax for these commands is defined in RFC 1939. After processing the quit command, the POP3 server enters the update phase and removes messages 1 and 2 from the mailbox. A problem with this download-and-delete mode is that the recipient, Bob, may be nomadic and may want to access his mail messages from multiple machines, for example, his office PC, his home PC, and his portable computer. The download-and-delete mode partitions Bob's mail messages over these three machines; in particular, if Bob first reads a message on his office PC, he will not be able to reread the message from his portable at home later in the evening. In the download-and-keep mode, the user agent leaves the messages on the mail server after downloading them. In this

case, Bob can reread messages from different machines; he can access a message from work and access it again later in the week from home. During a POP3 session between a user agent and the mail server, the POP3 server maintains some state information; in particular, it keeps track of which user messages have been marked deleted. However, the POP3 server does not carry state information across POP3 sessions. This lack of state information across sessions greatly simplifies the implementation of a POP3 server. IMAP With POP3 access, once Bob has downloaded his messages to the local machine, he can create mail folders and move the downloaded messages into the folders. Bob can then delete messages, move messages across folders, and search for messages (by sender name or subject). But this paradigm—namely, folders and messages in the local machine—poses a problem for the nomadic user, who would prefer to maintain a folder hierarchy on a remote server that can be accessed from any computer. This is not possible with POP3—the POP3 protocol does not provide any means for a user to create remote folders and assign messages to folders. To solve this and other problems, the IMAP protocol, defined in [RFC 3501], was invented. Like POP3, IMAP is a mail access protocol. It has many more features than POP3, but it is also significantly more complex. (And thus the client and server side implementations are significantly more complex.) An IMAP server will associate each message with a folder; when a message first arrives at the server, it is associated with the recipient's INBOX folder. The recipient can then move the message into a new, user-created folder, read the message, delete the message, and so on. The IMAP protocol provides commands to allow users to create folders and move messages from one folder to another. IMAP also provides commands that allow users to search remote folders for messages matching specific criteria. Note that, unlike POP3, an IMAP server maintains user state information across IMAP sessions—for example, the names of the folders and which messages are associated with which folders. Another important feature of IMAP is that it has commands that permit a user agent to obtain components of messages. For example, a user agent can obtain just the message header of a message or just one part of a multipart MIME message. This feature is useful when there is a low-bandwidth connection (for example, a slow-speed modem link) between the user agent and its mail server. With a low-bandwidth connection, the user may not want to download all of the messages in its mailbox, particularly avoiding long messages that might contain, for example, an audio or video clip.

Web-Based E-Mail More and more users today are sending and accessing their e-mail through their Web browsers. Hotmail introduced Web-based access in the mid 1990s. Now Web-based e-mail is also provided by Google, Yahoo!, as well as just about every major university and corporation. With this service, the user agent is an ordinary Web browser, and the user communicates with its remote mailbox via HTTP. When a recipient, such as Bob, wants to access a message in his mailbox, the e-mail message is sent from Bob's mail server to Bob's browser using the HTTP protocol rather than the POP3 or IMAP protocol. When a sender, such as Alice, wants to send an e-mail message, the e-mail message is sent from her browser to her mail server over HTTP rather than over SMTP. Alice's mail server, however, still sends messages to, and receives messages from, other mail servers using SMTP.

2.4 DNS—The Internet's Directory Service

We human beings can be identified in many ways. For example, we can be identified by the names that appear on our birth certificates. We can be identified by our social security numbers. We can be identified by our driver's license numbers. Although each of these identifiers can be used to identify people, within a given context one identifier may be more appropriate than another. For example, the computers at the IRS (the infamous tax-collecting agency in the United States) prefer to use fixed-length social security numbers rather than birth certificate names. On the other hand, ordinary people prefer the more mnemonic birth certificate names rather than social security numbers. (Indeed, can you imagine saying, "Hi. My name is 132-67-9875. Please meet my

husband, 178-87-1146.”) Just as humans can be identified in many ways, so too can Internet hosts. One identifier for a host is its hostname. Hostnames—such as `www.facebook.com`, `www.google.com`, `gaia.cs.umass.edu`—are mnemonic and are therefore appreciated by humans. However, hostnames provide little, if any, information about the location within the Internet of the host. (A hostname such as `www.eurecom.fr`, which ends with the country code `.fr`, tells us that the host is probably in France, but doesn’t say much more.) Furthermore, because hostnames can consist of variable-length alphanumeric characters, they would be difficult to process by routers. For these reasons, hosts are also identified by so-called IP addresses. We discuss IP addresses in some detail in Chapter 4, but it is useful to say a few brief words about them now. An IP address consists of four bytes and has a rigid hierarchical structure. An IP address looks like `121.7.106.83`, where each period separates one of the bytes expressed in decimal notation from 0 to 255. An IP address is hierarchical because as we scan the address from left to right, we obtain more and more specific information about where the host is located in the Internet (that is, within which network, in the network of networks). Similarly, when we scan a postal address from bottom to top, we obtain more and more specific information about where the addressee is located.

2.4.1 Services Provided by DNS

We have just seen that there are two ways to identify a host—by a hostname and by an IP address. People prefer the more mnemonic hostname identifier, while routers prefer fixed-length, hierarchically structured IP addresses. In order to reconcile these preferences, we need a directory service that translates hostnames to IP addresses. This is the main task of the Internet’s domain name system (DNS). The DNS is (1) a distributed database implemented in a hierarchy of DNS servers, and (2) an application-layer protocol that allows hosts to query the distributed database. The DNS servers are often UNIX machines running the Berkeley Internet Name Domain (BIND) software [BIND 2016]. The DNS protocol runs over UDP and uses port 53. DNS is commonly employed by other application-layer protocols—including HTTP and SMTP to translate user-supplied hostnames to IP addresses. As an example, consider what happens when a browser (that is, an HTTP client), running on some user’s host, requests the URL `www.someschool.edu/index.html`. In order for the user’s host to be able to send an HTTP request message to the Web server `www.someschool.edu`, the user’s host must first obtain the IP address of `www.someschool.edu`. This is done as follows.

1. The same user machine runs the client side of the DNS application.
2. The browser extracts the hostname, `www.someschool.edu`, from the URL and passes the hostname to the client side of the DNS application.
3. The DNS client sends a query containing the hostname to a DNS server.
4. The DNS client eventually receives a reply, which includes the IP address for the hostname.
5. Once the browser receives the IP address from DNS, it can initiate a TCP connection to the HTTP server process located at port 80 at that IP address.

We see from this example that DNS adds an additional delay—sometimes substantial—to the Internet applications that use it. Fortunately, as we discuss below, the desired IP address is often cached in a “nearby” DNS server, which helps to reduce DNS network traffic as well as the average DNS delay. DNS provides a few other important services in addition to translating hostnames to IP addresses:

Host aliasing. A host with a complicated hostname can have one or more alias names. For example, a hostname such as `relay1.west-coast.enterprise.com` could have, say, two aliases such as `enterprise.com` and `www.enterprise.com`. In this case, the hostname `relay1.west-coast.enterprise.com` is said to be a canonical hostname. Alias hostnames, when present, are typically more mnemonic than canonical hostnames. DNS can be invoked by an application to obtain the canonical hostname for a supplied alias hostname as well as the IP address of the host.

Mail server aliasing. For obvious reasons, it is highly desirable that e-mail addresses be mnemonic. For example, if Bob has an account with Yahoo Mail, Bob’s e-mail address might be

as simple as bob@yahoo.mail . However, the hostname of the Yahoo mail server is more complicated and much less mnemonic than simply yahoo.com (for example, the canonical hostname might be something like relay1.west-coast.yahoo.com). DNS can be invoked by a mail application to obtain the canonical hostname for a supplied alias hostname as well as the IP address of the host. In fact, the MX record (see below) permits a company's mail server and Web server to have identical (aliased) hostnames; for example, a company's Web server and mail server can both be called enterprise.com . Load distribution. DNS is also used to perform load distribution among replicated servers, such as replicated Web servers. Busy sites, such as cnn.com , are replicated over multiple servers, with each server running on a different end system and each having a different IP address. For replicated Web servers, a set of IP addresses is thus associated with one canonical hostname. The DNS database contains this set of IP addresses. When clients make a DNS query for a name mapped to a set of addresses, the server responds with the entire set of IP addresses, but rotates the ordering of the addresses within each reply. Because a client typically sends its HTTP request message to the IP address that is listed first in the set, DNS rotation distributes the traffic among the replicated servers. DNS rotation is also used for e-mail so that multiple mail servers can have the same alias name. Also, content distribution companies such as Akamai have used DNS in more sophisticated ways [Dilley 2002] to provide Web content distribution (see Section 2.6.3). The DNS is specified in RFC 1034 and RFC 1035, and updated in several additional RFCs. It is a complex system, and we only touch upon key aspects of its PRINCIPLES IN PRACTICE DNS: CRITICAL NETWORK FUNCTIONS VIA THE CLIENT-SERVER PARADIGM Like HTTP, FTP, and SMTP, the DNS protocol is an application-layer protocol since it (1) runs between communicating end systems using the client-server paradigm and (2) relies on an underlying end-to-end transport protocol to transfer DNS messages between communicating end systems. In another sense, however, the role of the DNS is quite different from Web, file transfer, and e-mail applications. Unlike these applications, the DNS is not an application with which a user directly interacts. Instead, the DNS provides a core Internet function—namely, translating hostnames to their underlying IP addresses, for user applications and other software in the Internet. We noted in Section 1.2 that much of the complexity in the Internet architecture is located at the “edges” of the network. The DNS, which implements the critical name-to-address translation process using clients and servers located at the edge of the network, is yet another example of that design philosophy. operation here. The interested reader is referred to these RFCs and the book by Albitz and Liu [Albitz 1993]; see also the retrospective paper [Mockapetris 1988], which provides a nice description of the what and why of DNS, and [Mockapetris 2005].

2.4.2 Overview of How DNS Works

We now present a high-level overview of how DNS works. Our discussion will focus on the hostname-to- IP-address translation service. Suppose that some application (such as a Web browser or a mail reader) running in a user's host needs to translate a hostname to an IP address. The application will invoke the client side of DNS, specifying the hostname that needs to be translated. (On many UNIX-based machines, gethostbyname() is the function call that an application calls in order to perform the translation.) DNS in the user's host then takes over, sending a query message into the network. All DNS query and reply messages are sent within UDP datagrams to port 53. After a delay, ranging from milliseconds to seconds, DNS in the user's host receives a DNS reply message that provides the desired mapping. This mapping is then passed to the invoking application. Thus, from the perspective of the invoking application in the user's host, DNS is a black box providing a simple, straightforward translation service. But in fact, the black box that implements the service is complex, consisting of a large number of DNS servers distributed around the globe, as well as an application-layer protocol that specifies how the DNS servers