CHAPTER 5

The Link Layer: Links, Access Networks, and LANs

In the previous chapter, 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 6.

5.1 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 6). 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 5.1, 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

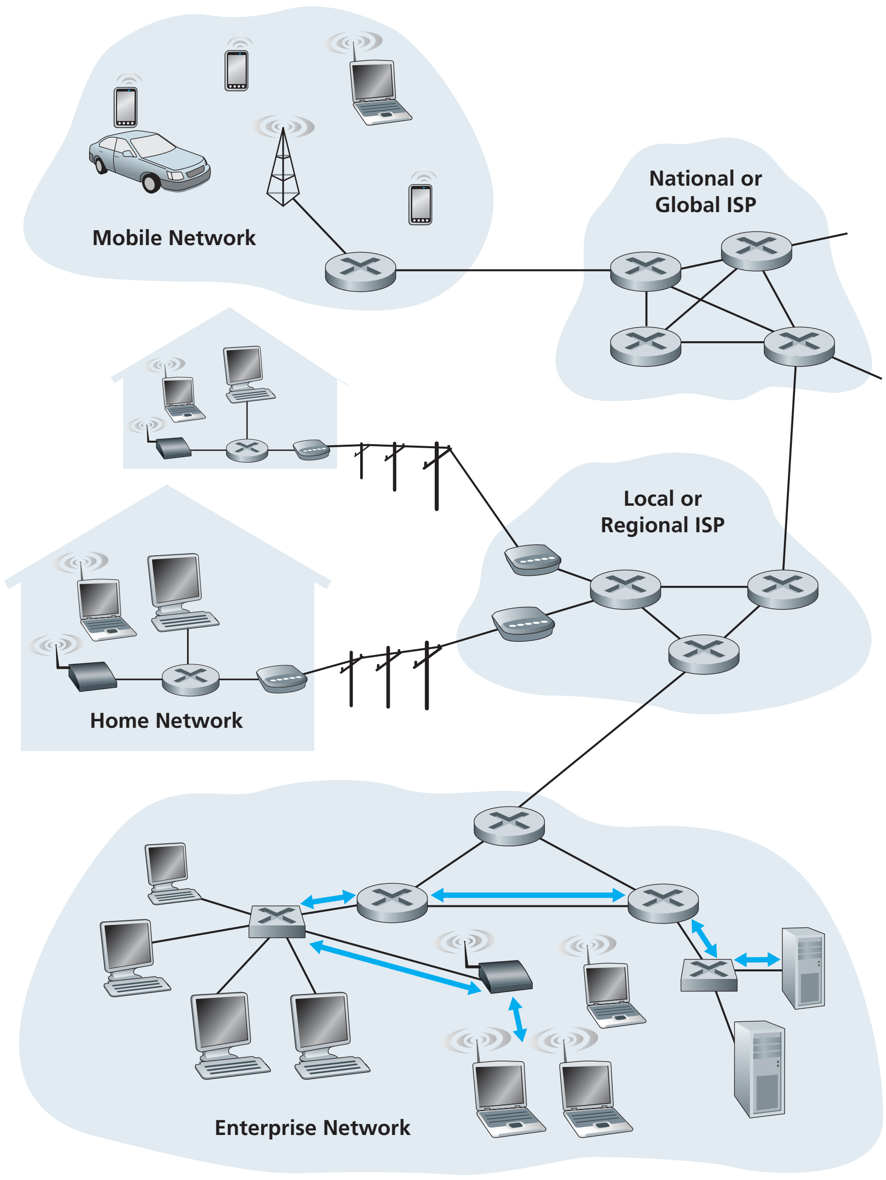


Figure 5.1 ♦ Six link-layer hops between wireless host and server

get the tourist from JFK to Geneva; and it is the responsibility 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.

5.1.1 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).

5.1.2 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 5.2 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 8254x controller [Intel 2012] implements the Ethernet protocols we'll study in Section 5.5; the Atheros AR5006 [Atheros 2012] controller implements the 802.11 WiFi protocols we'll study in Chapter 6. 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.

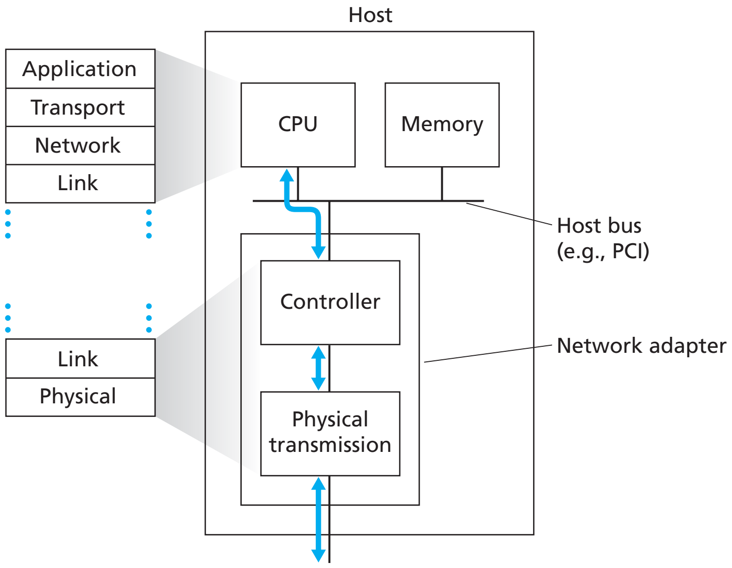


Figure 5.2 ♦ Network adapter: its relationship to other host components and to protocol stack functionality

Figure 5.2 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 components. Figure 5.2 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 [2012] provides a readable overview (as well as a detailed description) of the 8254x controller from a software-programming point of view.

5.2 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 5.3 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 5.3 (we ask whether an error is detected, not whether an error has occurred!) is important. Error-detection and -correction techniques

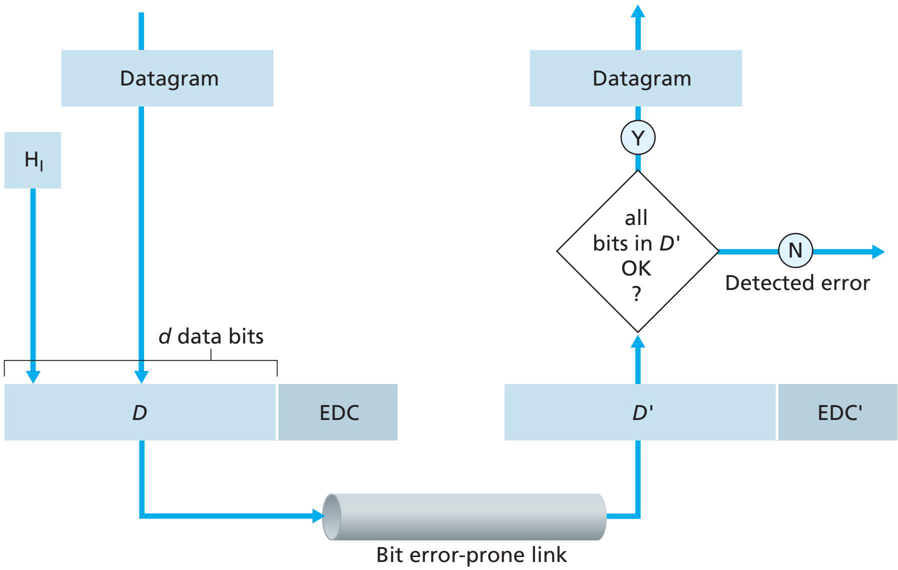


Figure 5.3 ♦ Error-detection and -correction scenario

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 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).

5.2.1 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 5.4, 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 5.4 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.

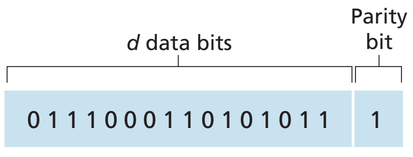


Figure 5.4 ♦ One-bit even parity

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

Figure 5.5 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 5.5 shows an example in

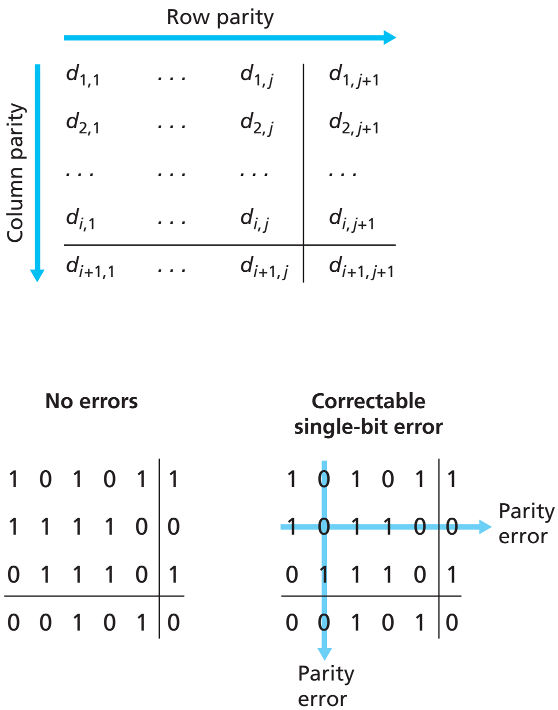


Figure 5.5 ♦ Two-dimensional even parity

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.

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].

5.2.2 Checksumming Methods

In checksumming techniques, the *d* bits of data in Figure 5.4 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.

5.2.3 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 5.6. 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
```

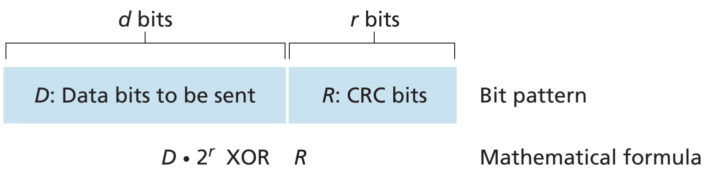


Figure 5.6 ♦ CRC

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 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 5.6. We'll use this algebraic characterization of the $d + r$ bit pattern from Figure 5.6 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 5.7 illustrates this calculation for the case of $D = 101110$, $d = 6$, $G = 1001$, and $r = 3$. The 9 bits transmitted in this case are 101110 011. You should check these calculations for yourself and also check that indeed $D \cdot 2^r = 101011 \cdot G$ XOR R.

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

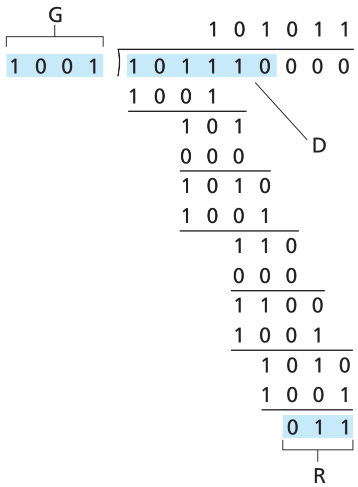


Figure 5.7 ♦ A sample CRC calculation

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.

5.3 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 that we'll cover later in this chapter. 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 also 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 5.8, 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 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

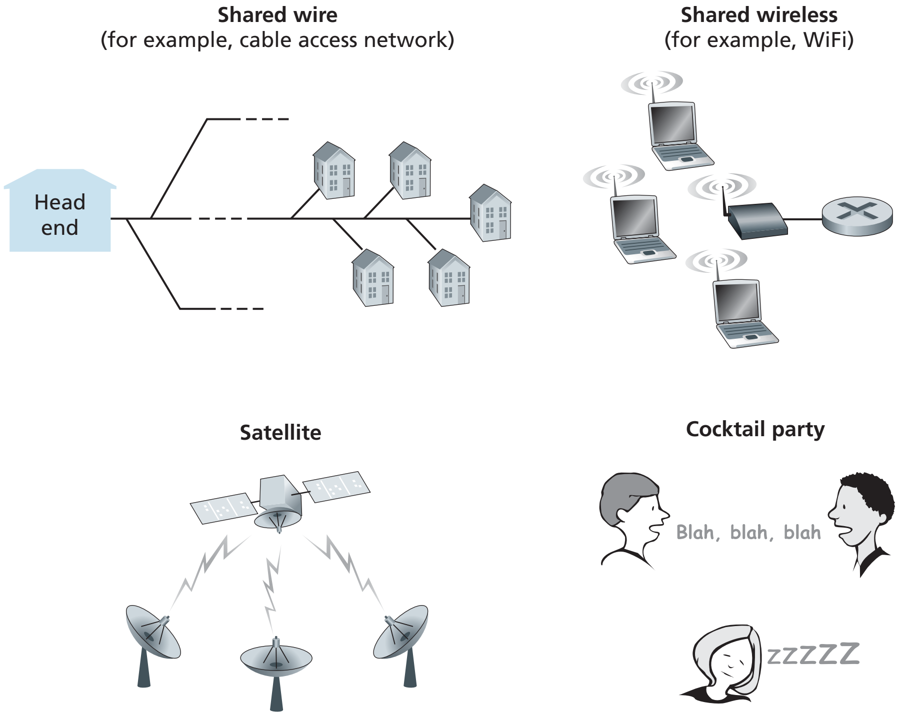


Figure 5.8 ♦ Various multiple access channels

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.

5.3.1 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 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 5.9 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.

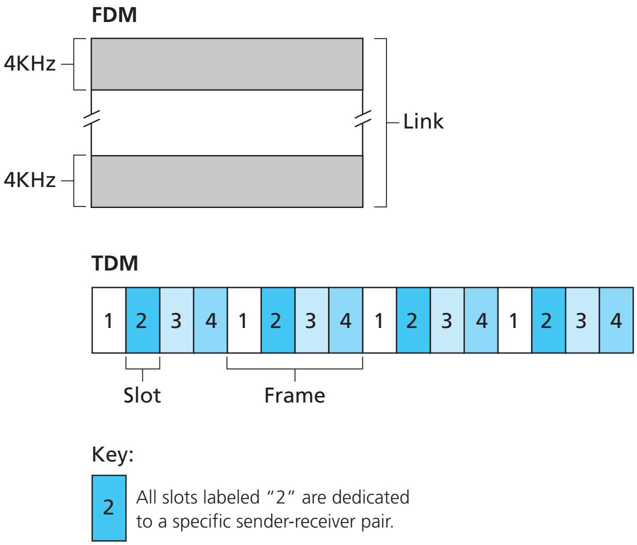


Figure 5.9 ♦ A four-node TDM and FDM example

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 6. 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.

5.3.2 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 concerns here. First, as shown in Figure 5.10, 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.

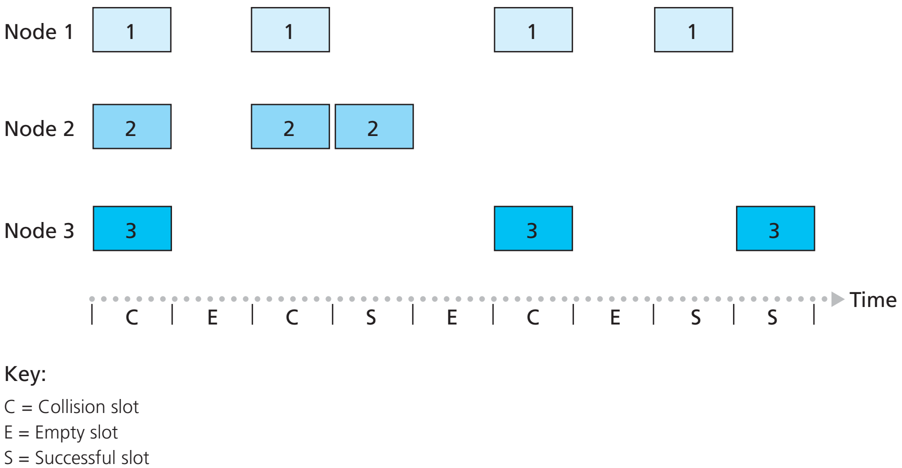


Figure 5.10 ♦ Nodes 1, 2, and 3 collide in the first slot. Node 2 finally succeeds in the fourth slot, node 1 in the eighth slot, and node 3 in the ninth slot

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 5.11, 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.

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

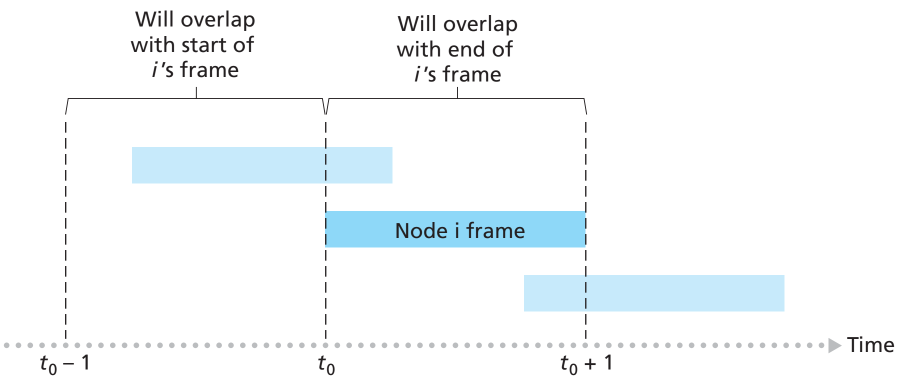


Figure 5.11 ♦ Interfering transmissions in pure ALOHA

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.

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 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 5.12 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.

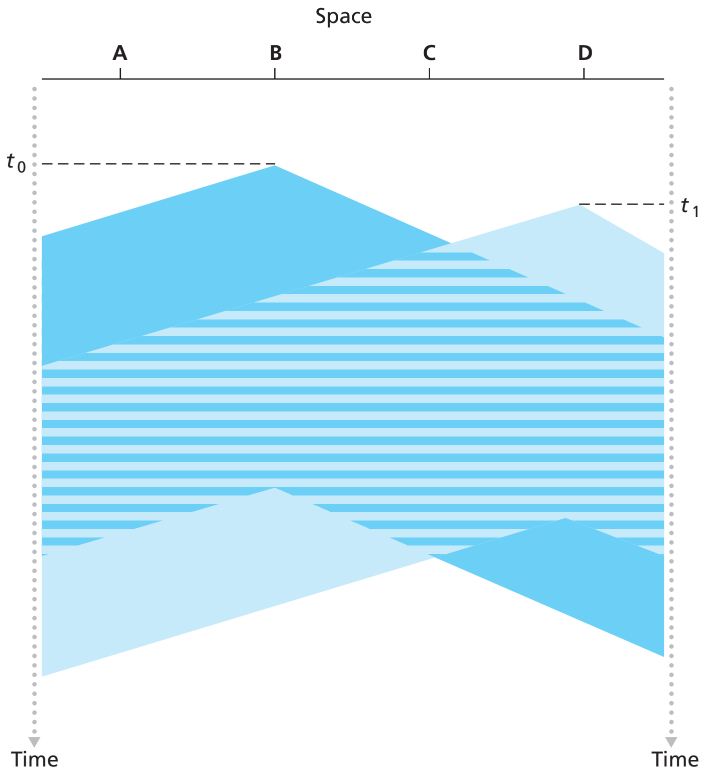


Figure 5.12 ♦ Space-time diagram of two CSMA nodes with colliding transmissions

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 5.12 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 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 5.12, 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 5.12, 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 5.13 shows the same scenario as in Figure 5.12, except that the two 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.

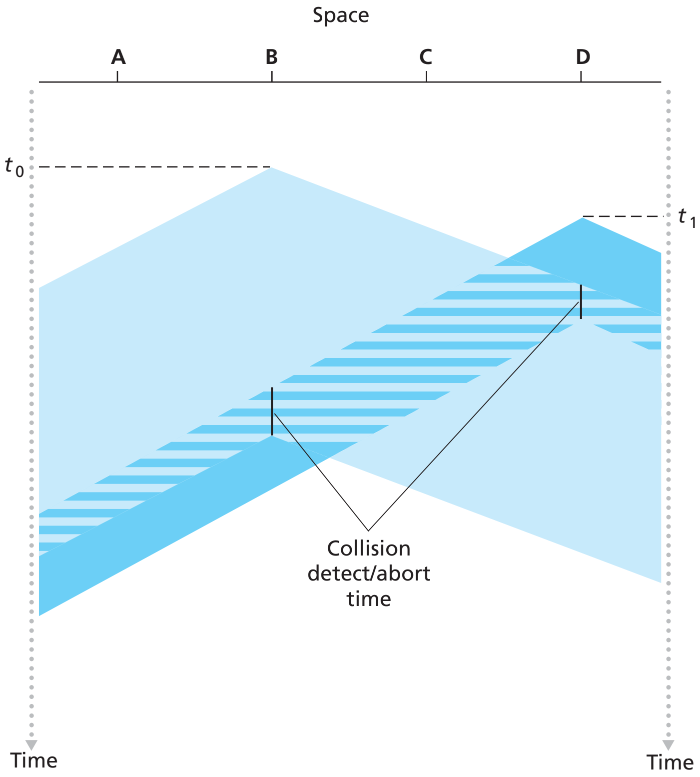


Figure 5.13 ♦ CSMA with collision detection

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., 0.01 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.

5.3.3 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 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.

6.3 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, including 802.11b, 802.11a, and 802.11g. Table 6.1 summarizes the main characteristics of these standards. 802.11g is by far the most popular technology. A number of dual-mode (802.11a/g) and tri-mode (802.11a/b/g) devices are also available.

The three 802.11 standards share many 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 all three standards allow for both "infrastructure mode" and "ad hoc mode," as we'll also shortly discuss. However, as shown in Table 6.1, the three standards have some major differences at the physical layer.

The 802.11b wireless LAN has a data rate of 11 Mbps and operates in the unlicensed frequency band of 2.4–2.485 GHz, competing for frequency spectrum with 2.4 GHz phones and microwave ovens. 802.1la wireless LANs can run at

Standard	Frequency Range (United States)	Data Rate
802.11b	2.4–2.485 GHz	up to 11 Mbps
802.11a	5.1–5.8 GHz	up to 54 Mbps
802.11g	2.4–2.485 GHz	up to 54 Mbps

Table 6.1 ♦ Summary of IEEE 802.11 standards

significantly higher bit rates, but do so at higher frequencies. By operating at a higher frequency, 802.11a LANs have a shorter transmission distance for a given power level and suffer more from multipath propagation. 802.11g LANs, operating in the same lower-frequency band as 802.11b and being backwards compatible with 802.11b (so one can upgrade 802.11b clients incrementally) yet with the higher-speed transmission rates of 802.11a, allows users to have their cake and eat it too.

A relatively new WiFi standard, 802.11n [IEEE 802.11n 2012], 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]. Depending on the modulation scheme used, transmission rates of several hundred megabits per second are possible with 802.11n.

6.3.1 The 802.11 Architecture

Figure 6.7 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

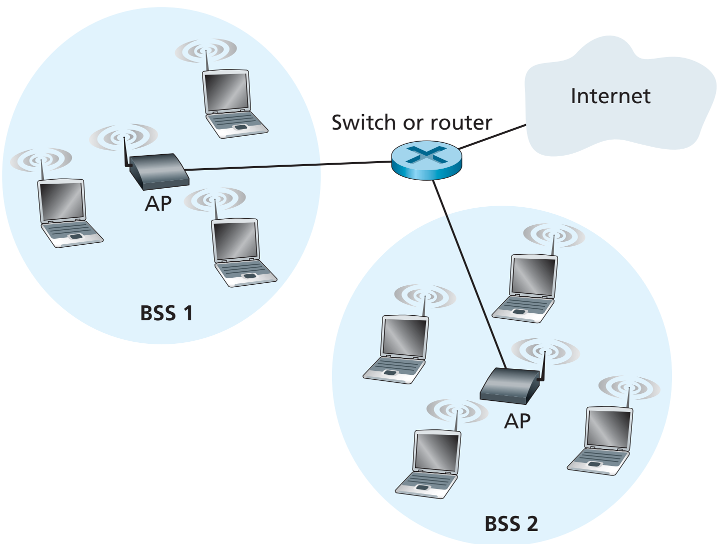


Figure 6.7 ♦ IEEE 802.11 LAN architecture

base station, known as an **access point (AP)** in 802.11 parlance. Figure 6.7 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.

As noted in Section 6.1, 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 6.8 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.

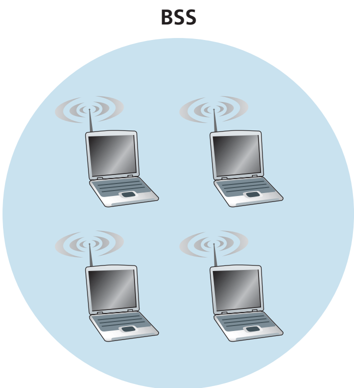


Figure 6.8 ♦ An IEEE 802.11 ad hoc network

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 "view available networks" in Microsoft Windows XP, 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 2.485 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 portable computer, seeking wireless Internet access and a blueberry muffin. Suppose there are five APs in the WiFi jungle. To gain Internet access, your wireless station needs to join exactly one of the subnets and hence needs to **associate** with exactly one of the APs. Associating means the wireless station 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 station, and your wireless station will send data frames into the Internet only through the associated AP. But how does your wireless station associate with a particular AP? And more fundamentally, how does your wireless station know which APs, if any, are out there in the jungle?

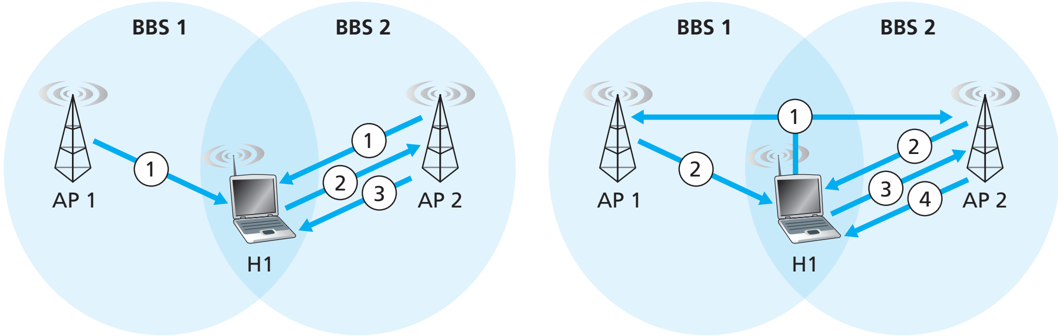
The 802.11 standard requires that an AP periodically send **beacon frames**, each of which includes the AP's SSID and MAC address. Your wireless station, knowing that APs are sending out beacon frames, scans the 11 channels, seeking beacon frames from any APs that may be out there (some of which may 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 host) 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 host. Typically, the host chooses the AP whose beacon frame is received with the highest signal strength. While a high signal strength is good (see, e.g., Figure 6.3), signal strength is not the only AP characteristic that will determine the performance a host receives. In particular, it's possible that the selected AP may have a strong signal, but may be overloaded with other affiliated hosts (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].

The process of scanning channels and listening for beacon frames is known as **passive scanning** (see Figure 6.9a). A wireless host can also perform **active scanning**, by broadcasting a probe frame that will be received by all APs within the wireless host's range, as shown in Figure 6.9b. APs respond to the probe request frame with a probe response frame. The wireless host can then choose the AP with which to associate from among the responding APs.

After selecting the AP with which to associate, the wireless host 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 host 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 host will want to join the subnet (in the IP addressing sense of Section 4.4.2) to which the AP belongs. Thus, the host 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 host simply as another host with an IP address in that subnet.

In order to create an association with a particular AP, the wireless station 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 station'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 end-point station 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



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 6.9 ♦ Active and passive scanning for access points

in Section 8.8 that the new IEEE 802.11i protocol defining security aspects of the 802.11 protocol family takes precisely this approach.

6.3.2 The 802.11 MAC Protocol

Once a wireless station is associated with an AP, it can start sending and receiving data frames to and from the access point. But because multiple stations may want to transmit data frames at the same time over the same channel, a multiple access protocol is needed to coordinate the transmissions. Here, a **station** is either a wireless station or an AP. As discussed in Chapter 5 and Section 6.2.1, 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 5.3.2 and 5.4.2 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 6.2.

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 6.2 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 6.10, 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 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 station 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 6.10.
2. Otherwise, the station chooses a random backoff value using binary exponential backoff (as we encountered in Section 5.3.2) and counts down this value 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.

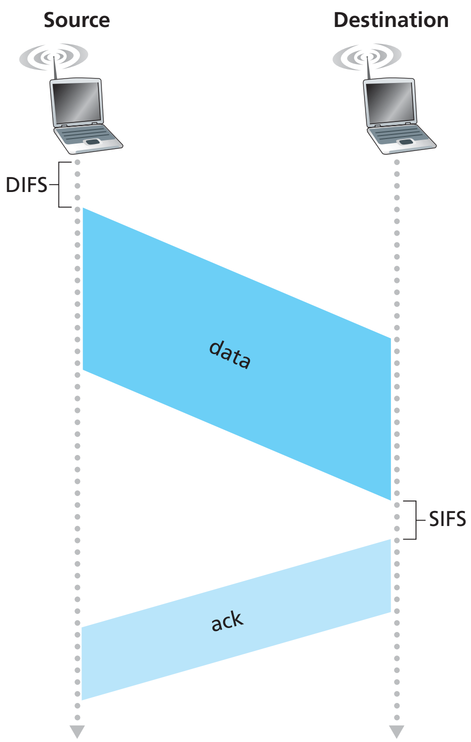


Figure 6.10 ♦ 802.11 uses link-layer acknowledgments

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 5.3.2), 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 5.12.

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 6.11, 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

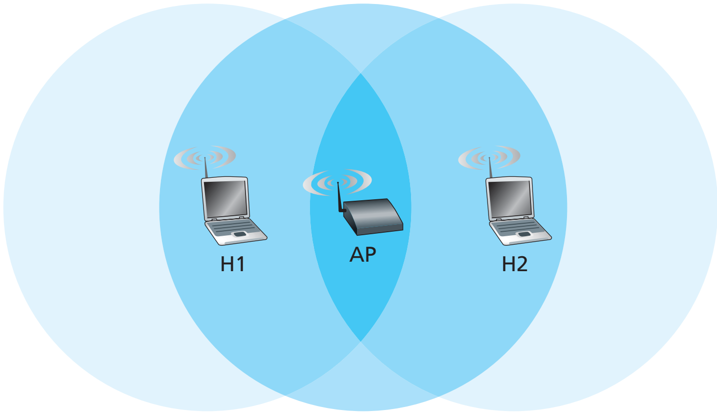


Figure 6.11 ♦ Hidden terminal example: H1 is hidden from H2, and vice versa

interiors of the shaded circles shown in Figure 6.11. 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 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 6.12, 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 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 6.12.

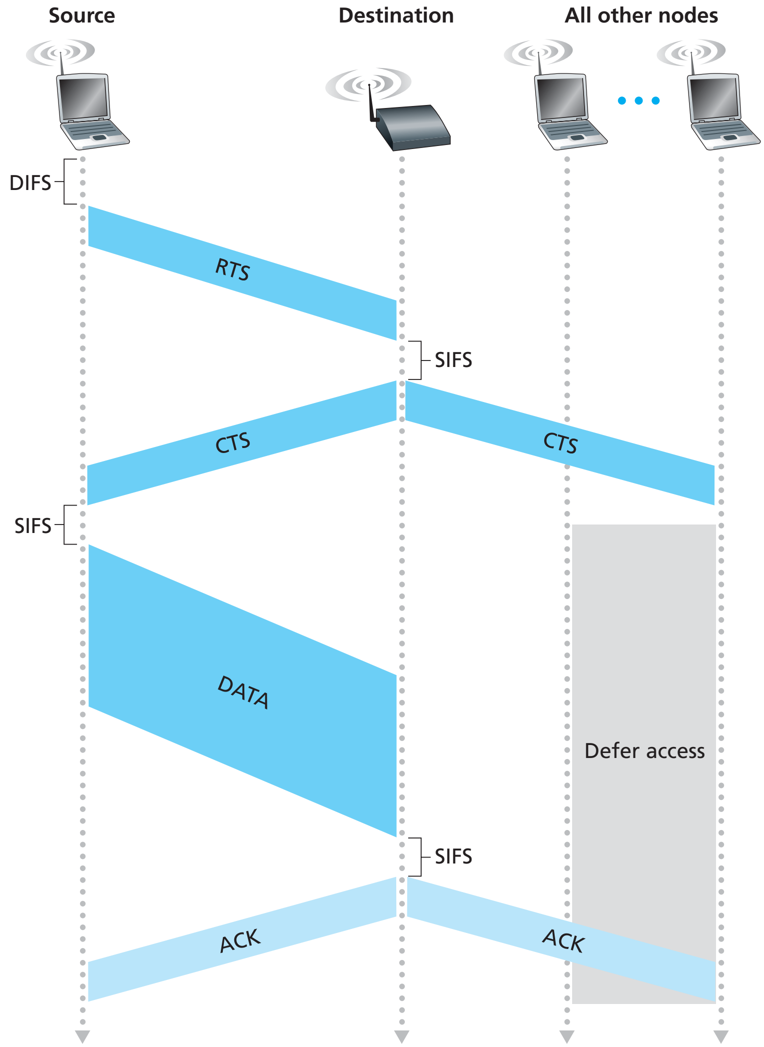


Figure 6.12 ♦ Collision avoidance using the RTS and CTS frames

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 companion 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 such a multi-hop wireless network operating in the rural Ganges plains in India that contains point-to-point 802.11 links.

6.3.3 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 6.13. 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.

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