CHAPTER 6

The Link Layer and LANs

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

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

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

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

6.1 Introduction to the Link Layer

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

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

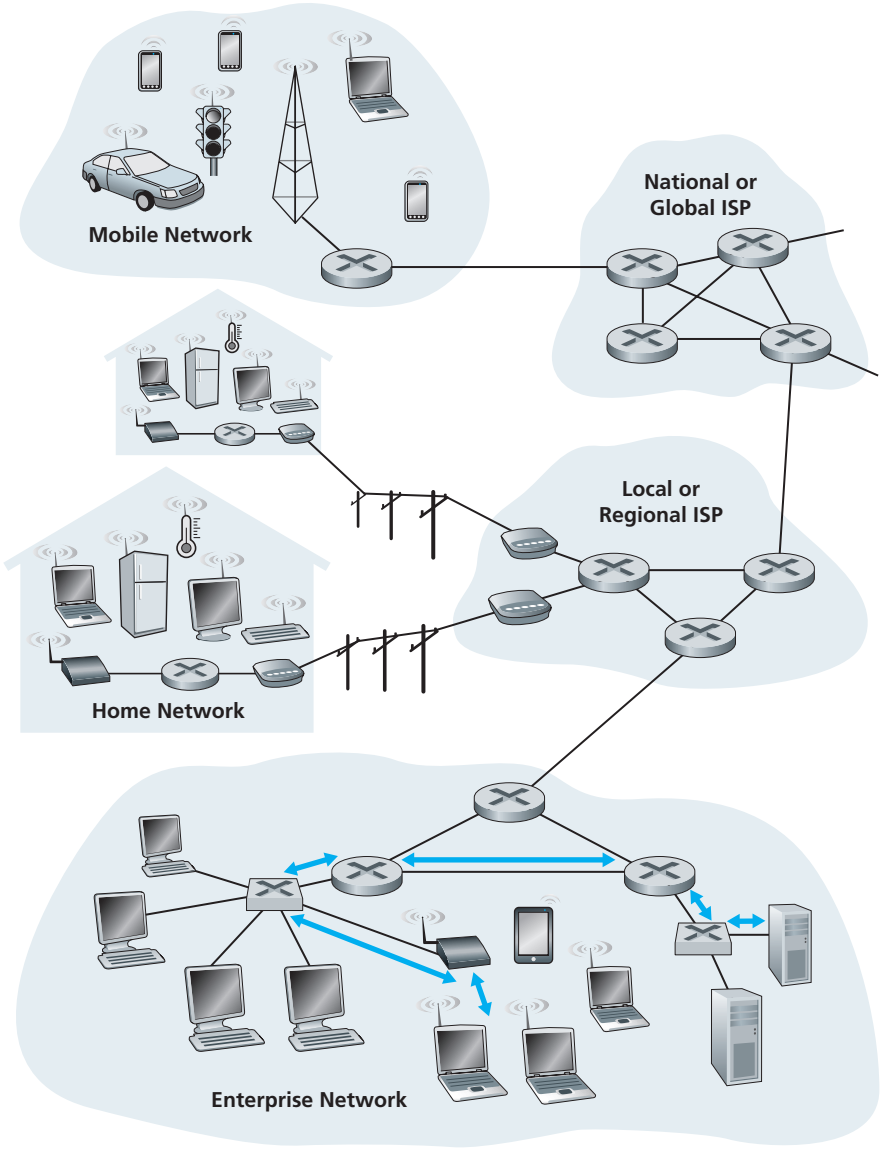

Figure 6.1 ♦ Six link-layer hops between wireless host and server

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

6.1.1 The Services Provided by the Link Layer

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

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

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

6.1.2 Where Is the Link Layer Implemented?

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

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

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

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

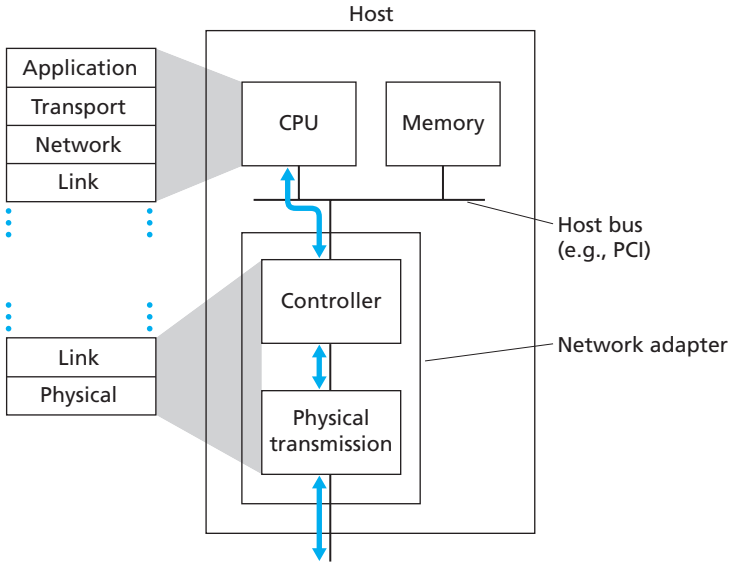

Figure 6.2 ♦ Network adapter: Its relationship to other host components and to protocol stack functionality

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

6.2 Error-Detection and -Correction Techniques

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

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

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

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

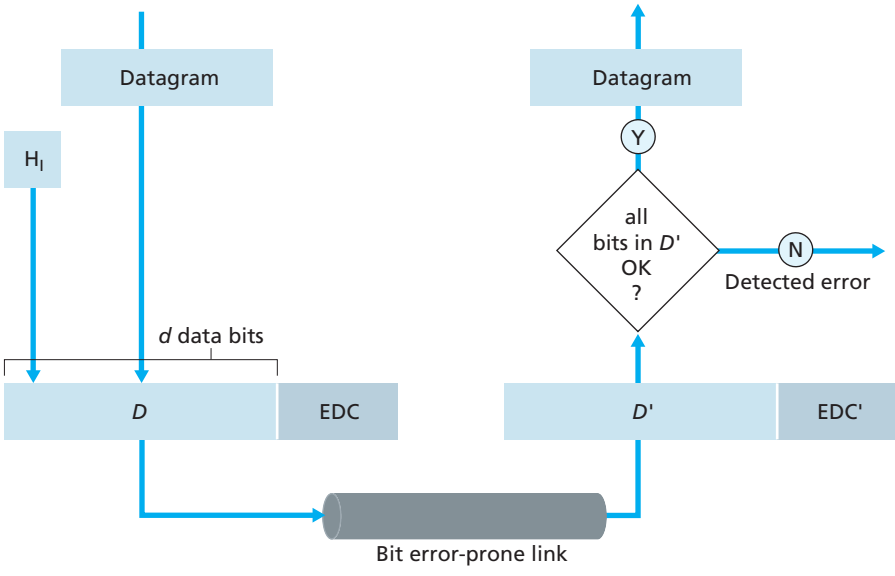

Figure 6.3 ♦ Error-detection and -correction scenario

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

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

6.2.1 Parity Checks

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

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

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

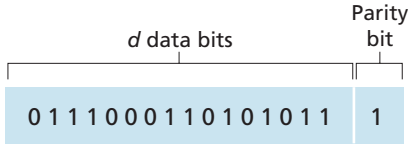

Figure 6.4 ♦ One-bit even parity

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

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

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

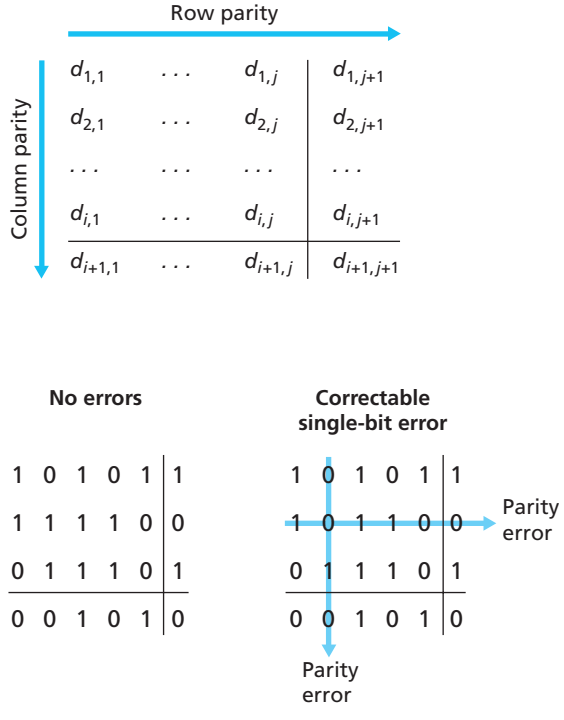

Figure 6.5 ♦ Two-dimensional even parity

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

6.2.2 Checksumming Methods

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

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

6.2.3 Cyclic Redundancy Check (CRC)

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

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

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

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

Also, we similarly have

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

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

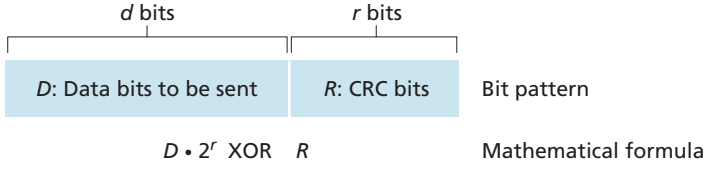

Figure 6.6 ♦ CRC

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

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

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

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

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

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

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

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

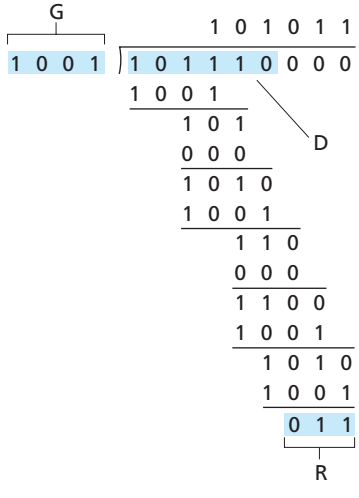

Figure 6.7 ♦ A sample CRC calculation

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

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

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

6.3 Multiple Access Links and Protocols

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

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