Computer Networking

A Top-Down Approach Featuring the Internet

James F. Kurose and Keith W. Ross

5.1 The Data Link Layer: Introduction, Services

In the previous chapter we learned that the network layer provides a communication service between two hosts. As shown in Figure 5.1-1, this communication path starts at the source host, passes through a series of routers, and ends at the destination host. We'll find it convenient here to refer to the hosts and the routers simply as **nodes (**since, as we'll see shortly, we will not be particularly concerned whether a node is a router or a host), and to the communication channels that connect adjacent nodes along the communication path as **links**. In order to move a datagram from source host to destination host, the datagram must be moved over each of the *individual links* in the path. In this chapter, we focus on the **data link layer,** which is responsible for transferring a datagram across an individual link. We'll first identify and study the services provided by the link layer. In sections 5.2 through 5.4, we'll then examine important principles behind the protocols that provide these services (including the topics of error detection and correction, so-called multiple access protocols that are used share a single physical link among multiple nodes, and link-level addressing). We'll see that many different types of link-level technology can be used to connect two nodes. In sections 5.5 through 5.10, we'll examine specific link-level architectures and protocols in more detail.

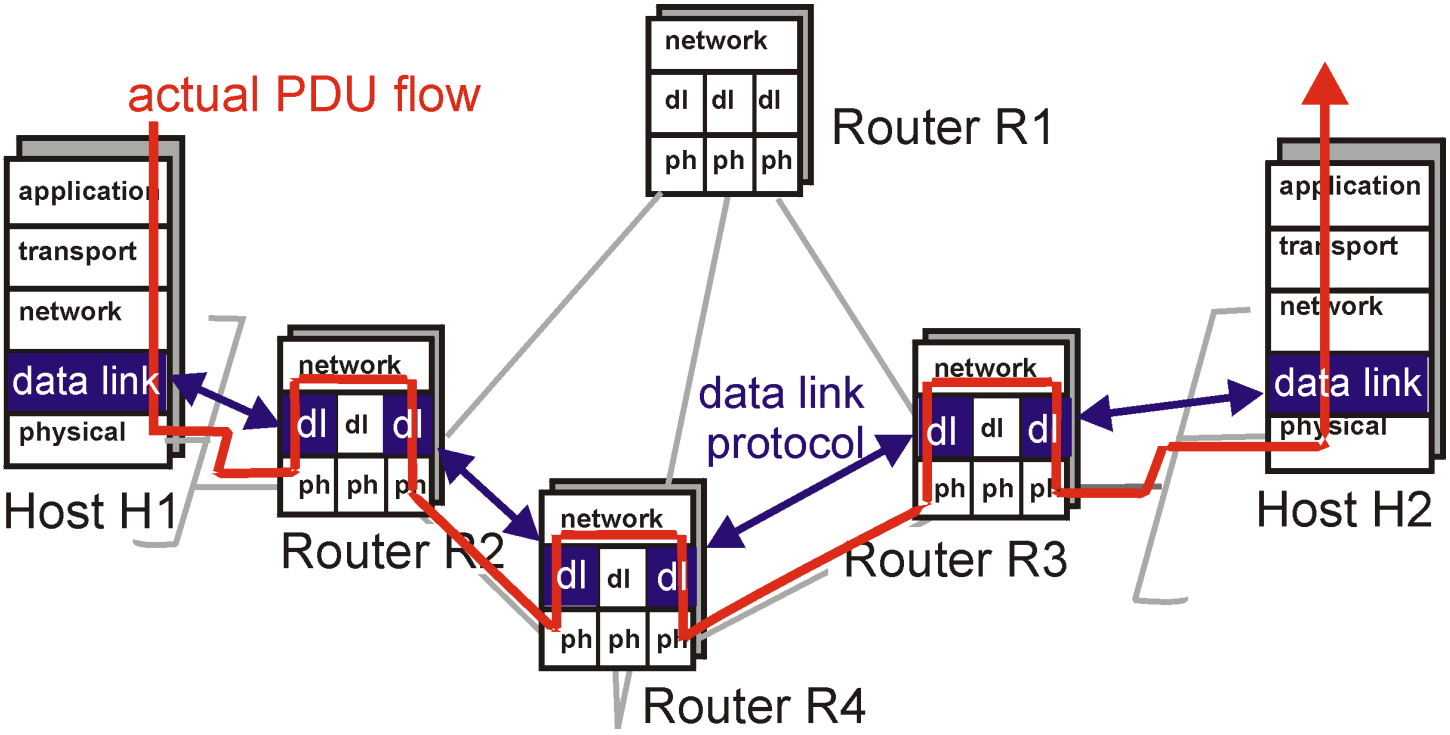

Figure 5.1-1: TheData Link Layer

5.1.1 The Services Provided by the Link Layer

A link-layer protocol is used to move a datagram over an individual link. The **link-layer protocol** defines the format of the packets exchanged between the nodes at the ends of the link, as well as the actions taken by these nodes when sending and receiving packets. Recall from Chapter 1 that the packets exchanged by a link-layer protocol are called **frames**, and that each link-layer frame typically encapsulates one network-layer datagram. As we shall see shortly, the

actions taken by a link-layer protocol when sending and receiving frames include error detection, retransmission, flow control and random access. Examples of link-layer protocols include Ethernet, token ring, FDDI, and PPP; in some contexts, ATM and frame relay can be considered link-layer protocols as well. We will cover these protocols in detail in the latter half of this chapter.

Whereas the network layer has the end-to-end job of moving transport-layer segments from the source host to the destination host, a link-layer protocol has the node-to-node job of moving a network-layer datagram over a *single link* in the path. An important characteristic of the link layer is that a datagram may be handled by different link-layer protocols on the different links in the path. For example, a datagram may be handled by Ethernet on the first link, PPP on the last link, and frame relay on all intermediate links. It is important to note that the services provided by the different link-layer protocols may be different. For example, a link-layer protocol may or may not provide reliable delivery. Thus, the network layer must be able to accomplish its end-to-end job in the face of a varying set of individual link-layer services.

In order to gain insight to 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 tr for a tourist traveling from Princeton, New Jersey to Lausanne, Switzerland. Suppose 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 airport, and finally a train from Geneva to Lausanne's train station. (There is a train station at Geneva's airport.) 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 responsibility of the Swiss train service to get the tourist from the 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. This *service* is used by the travel agent to plan the tourist's trip. In this transportation analogy, the tourist is analogous to a datagram, each transportation segment is analogous to a communication link, the transportation mode is analogous to the link-layer protocol, and the travel agent who plans the trip is analogous to a routing protocol.

The basic service of the link layer is to "move" a datagram from one node to an adjacent node over a single communication link. But the details of the link-layer service depend on the specific link-layer protocol that is employed over the link. Possible services that can be offered by a link-layer protocol include:

- **Framing and link access:** Almost all link-layer protocols encapsulate each network-layer datagram within a link-layer frame before transmission onto the link. A frame consists of a data field, in which the network-layer datagram is inserted, and a number of header fields. (A frame may also include trailer fields; however, we will refer to both header and trailer fields as header fields.) A data link protocol specifies the structure of the frame, as well as a channel access protocol that specifies the rules by which a frame is transmitted onto the link. For point-to-point links that have a single sender on one end of the link and a single receiver at the other end of the link, the link access protocol is simple (or non-existent) - 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 channel access protocol serves to coordinate the frame transmissions of the many nodes; we cover multiple access protocols in detail in section 5.3 . We'll see several different frame formats when we examine specific link-layer protocols in the second half of this chapter. In section 5.3, we'll see that frame headers also often include fields for a node's so-called **physical address**, which is completely *distinct* from the node's network layer (e.g., IP) address.

- **Reliable delivery:** If a link-layer protocol provides the reliable-delivery service, then it guarantees to move

each network-layer datagram across the link without error. Recall that transport-layer protocols (such as TCP) may also provide a reliable-delivery service. Similar to a transport-layer reliable-delivery service, a link-layer reliable-delivery service is 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 at which the error occurs, rather than forcing an end-to-end retransmission of the data by transport- or application-layer protocol. However, link-layer reliable delivery is often considered to be unnecessary overhead for low bit-error links, including fiber, coax and many twisted-pair copper links. For this reason, many of the most popular link-layer protocols do not provide a reliable-delivery service.

- **Flow control:** The nodes on each side of a link have a limited amount of packet buffering capacity. This is a potential problem, as a receiving node may receive frames at a rate faster than it can process the frames (over some time interval). Without flow control, the receiver's buffer can overflow and frames can get lost. Similar to the transport layer, a link-layer protocol can provide flow control in order to prevent the sending node on one side of a link from overwhelming the receiving node on the other side of the link.

- **Error detection:** A node's receiver can incorrectly decide that a bit in a frame to be a zero when it was transmitted as a one (and vice versa). These 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 for a node to detect the presence of one or more errors. This is done by having the transmitting node set error detection bits in the frame, and having the receiving node perform an error check. Error detection is a very common service among link-layer protocols. Recall from Chapters 3 and 4 that the transport layer and network layers in the Internet also provide a limited form of error detection. Error detection in the link layer is usually more sophisticated and implemented in hardware.

- **Error correction:** Error correction is similar to error detection, except that a receiver can not only detect whether errors have been introduced in the frame but can also determine exactly where in the frame the errors have occurred (and hence correct these errors). Some protocols (such as ATM) provide link-layer error correction for the packet header rather than for the entire packet. We cover error detection and correction in section 5.2.

- **Half-Duplex and Full-Dulpex:** With full-duplex transmission, both nodes at the ends of a link may transmit packets at the same time. With half-duplex transmission, a node cannot both transmit and receive at the same time.

As noted above, many of the services provided by the link layer have strong parallels with services provided at the transport layer. For example, both the link layer and the transport layer can provide reliable delivery. Although the mechanisms used to provide reliable delivery in the two layers are similar (see Section 3.4), the two reliable delivery services are not the same. A transport protocol provides reliable delivery between two processes on an end-to-end basis; a reliable link-layer protocol provides the reliable-delivery service between two nodes connected by a single link. Similarly, both link-layer and transport-layer protocols can provide flow control and error detection; again, flow control in a transport-layer protocol is provided on an end-to-end basis, whereas it is provided in a link-layer protocol on a node-to-adjacent-node basis.

5.1.2 Adapters Communicating

For a given communication link, the link-layer protocol is for the most part implemented in a pair of **adapters**. An adapter is a board (or a PCMCIA card) that typically contains RAM, DSP chips, a host bus interface and a link

interface. Adapters are also commonly known as *network interface cards* or *NICs*. As shown in Figure 5.1-2, the network layer in the transmitting node (i.e., a host or router) passes a network-layer datagram to the adapter that handles the sending side of the communication link. The adapter encapsulates the datagram in a frame and then transmits the frame into the communication link. At the other side, the receiving adapter receives the entire frame, extracts the network-layer datagram, and passes it to the network layer. If the link-layer protocol provides error detection, then it is the sending adapter that sets the error detection bits and it is the receiving adapter that performs the error checking. If the link-layer protocol provides reliable delivery, then the mechanisms for reliable delivery (e.g., sequence numbers, timers and acknowledgments) are entirely implemented in the adapters. If the link-layer protocol provides random access (see Section 5.3), then the random access protocol is entirely implemented in the adapters.

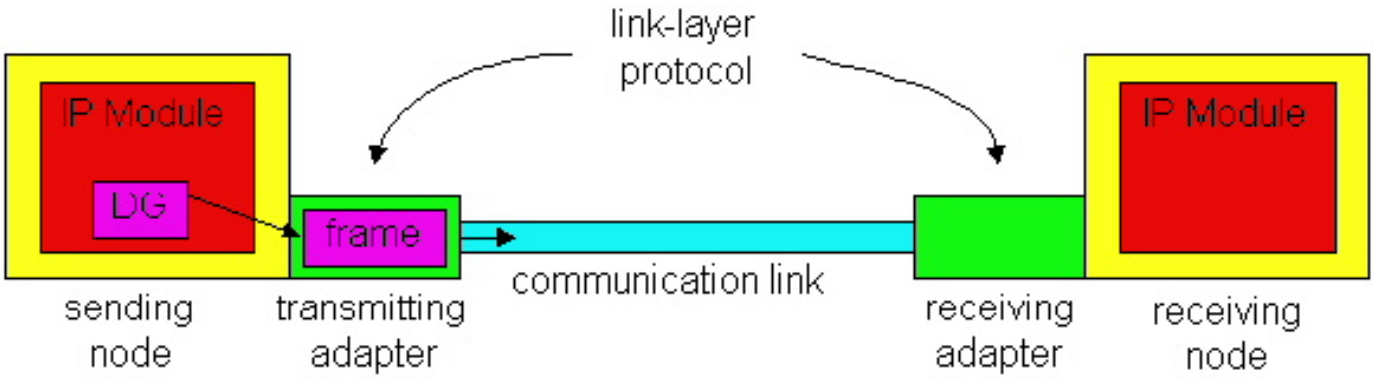

Figure 5.1-2: The link-layer protocol for a communication link is implemented in the adapters at the two ends of the link. DG abbreviates "datagram".

A computer in itself, an adapter is a semi-autonomous unit. For example, an adapter can receive a frame, determine if a frame is in error and discard the frame without notifying its "parent" node. An adapter that receives a frame only interrupts its parent node when it wants to pass a network-layer datagram up the protocol stack. Similarly, when a node passes a datagram down the protocol stack to an adapter, the node fully delegates to the adapter the task of transmitting the datagram across that link. On the other hand, an adapter is not an completely autonomous unit. Although we have shown the adapter as a separate "box" in Figure 5.3.1, the adapter is typically housed in the same physical box as rest of the node, shares power and busses with the rest of the node, and is ultimately under the control of the node.

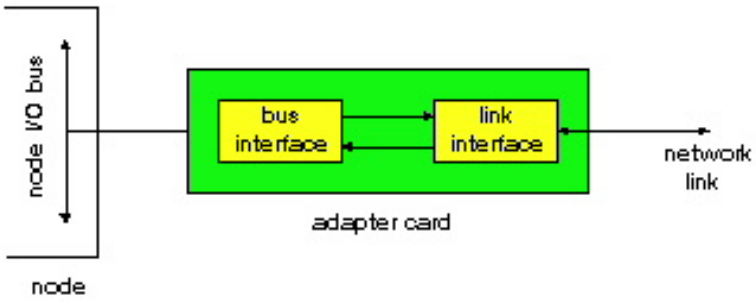

Figure 5.1-3: The adapter is a semi-autonomous unit.

As shown in Figure 5.1.3, the main components of an adapter are the bus interface and the link interface. The bus interface is responsible for communicating with the adapter's parent node. It sends to and receives from the parent node network-layer datagrams and control information. The link interface is responsible for implementing the link-layer protocol. In addition to framing and de-framing datagrams, it may provide error detection, random access and other link-layer functions. It also includes the transmit and receive circuitry. For popular link-layer technologies, such as

Ethernet, the link interface is implemented by chip set that can be bought on the commodity market. For this reason, Ethernet adapters are incredibly cheap -- often less than $30 for 10 Mbps and 100 Mbps transmission rates.

Adapter design has become very sophisticated over the years. One of the critical issues in adapter performance has always been whether the adapter can move data in and out of a node at the full line speed, that is, at the transmission rate of the link. You can learn more about adapter architecture for 10Mbps Ethernet, 100 Mbps Ethernet and 155 Mbps ATM by visiting the 3Com adapter page [3Com]. Data Communications magazine provides a nice introduction to Gbps Ethernet adapters [GigaAdapter].

References

[3Com] 3Com Corporation, Network Interface Cards, http://www.3com.com/products/nics.html
[GigaAdapter] Data Communications, "Lan Gear," http://www.data.com/hot_products/lan_gear/alteon.html

Return to Table of Contents

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 data-link-layer frame sent from one node to another physically-connected neighboring node - are two services often provided by the data 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 (e.g., [Schwartz 1980]), 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 data link layer.

Figure 5.2-1 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 data 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*' are received. Note that *D*' and *EDC*' may differ from the original *D* and *EDC* as a result of in-transit bit flips.

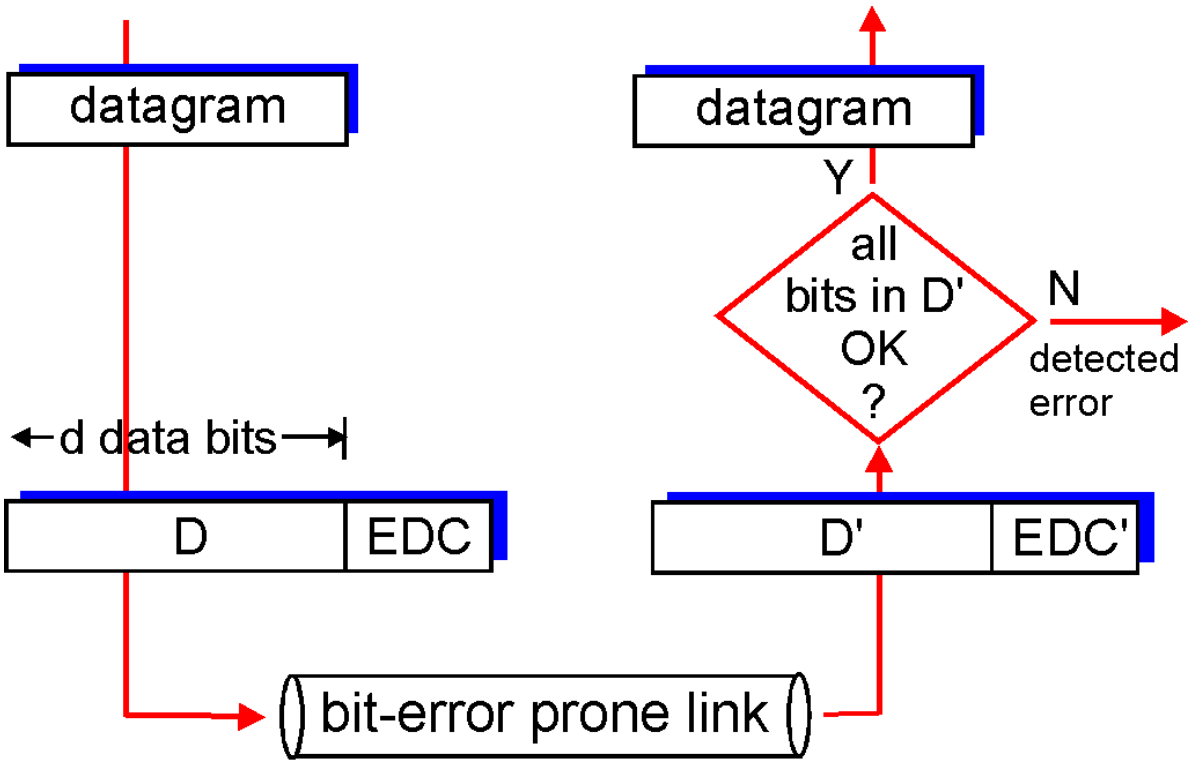

Figure 5.2-1: Error detection and correction scenario

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 5.2-1 (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. That is, even with the use of error detection bits there will still be a possibility that **undetected bit errors** will occur, i.e., that the receiver will 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 some other field in the frame's header have been corrupted. We thus want to choose an error detection scheme so that the probability of such occurrences is small. Generally, more sophisticated error detection and correction techniques (i.e., those that have a smaller probability of allowing undetected bit errors) incur a larger overhead - more computation is need 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 employed in the transport layer) and cyclic redundancy checks (which are typically employed in the data link layer).

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 5.2-1, 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 1's 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 are an odd number of 1's. Figure 5.2-2 illustrates an even parity scheme, with the single parity bit being stored in a separate field.

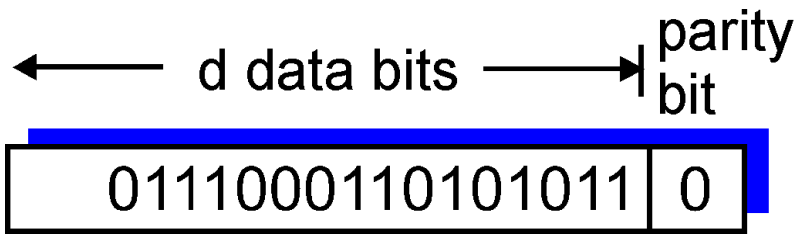

Figure 5.2-2: One-bit even parity

Receiver operation is also simple with a single parity bit. The receiver need only count the number of 1's 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-partity 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 cosider a simple generalization of one-bit parity that will provide us with insight into error correction techniques.

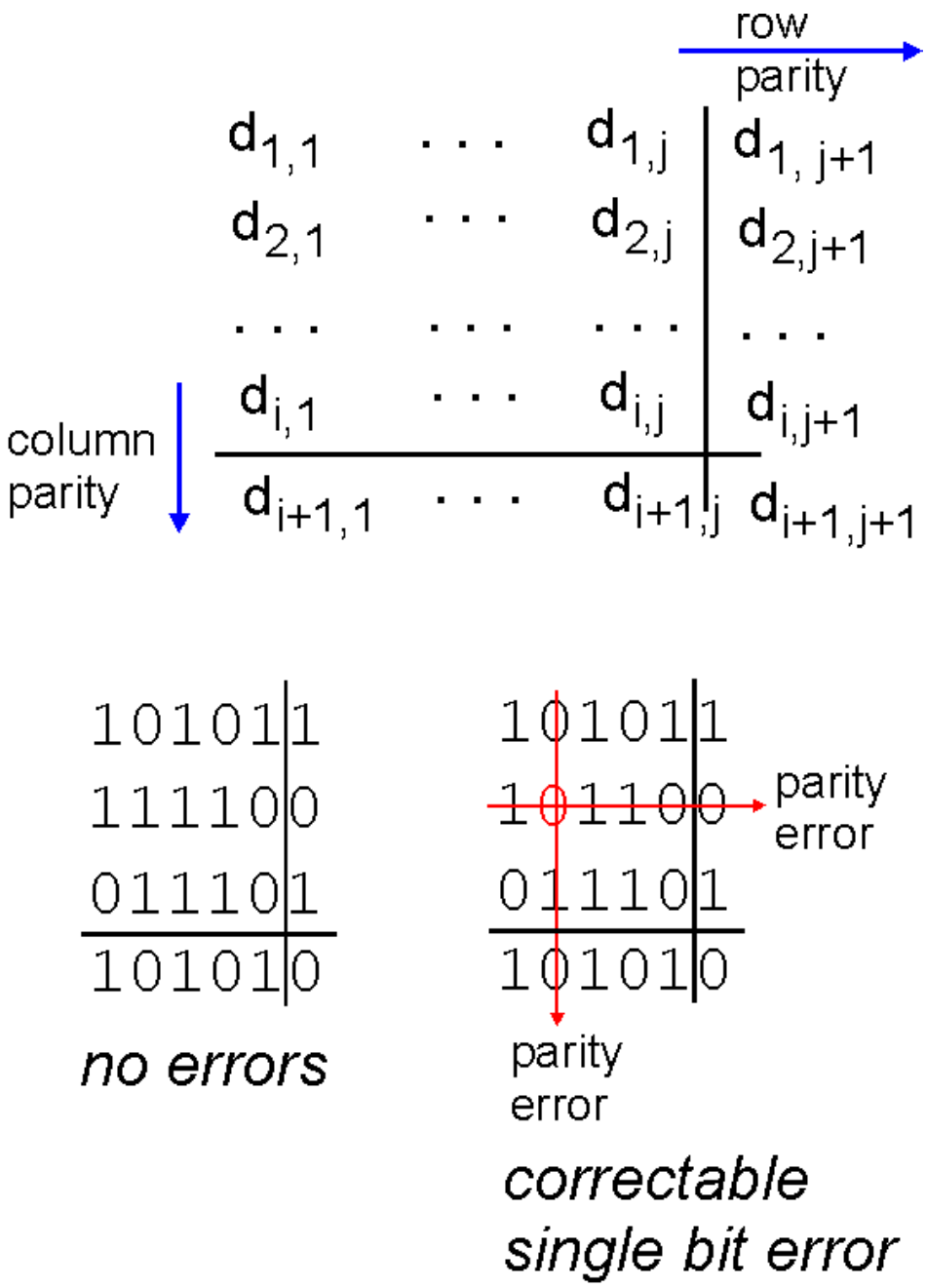

Figure 5.2-3: Two-dimensional even parity

Figure 5.2-3 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 are the data link 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 5.2-3 shows an example in which the 0-valued bit in position (1,1) is corrupted and switched to a 1 -- an error that is both detectable and correctable at the receiver. Although our discussion has focussed 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 CD's. In a network setting, FEC techniques can be used by themselves, or in conjunction with the ARQ techniques we examined in Chapter 3. FEC techniques are valuable because they can decrease the number of sender retransmissions required. Perhaps more importantly, they allow for immediate correction of errors at the receiver. This avoids having to wait 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]. Recent work examining the use of FEC in error control protocols include [Biersack 1992, Nonnenmacher 1998, Byers 1998, Shacham 1990].

5.2.2 Checksumming Methods

In checksumming techniques, the *d* bits of data in Figure 5.2-1 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 so-called **Internet checksum** [RFC 1071] is based on this approach -- bytes of data are treated as 16-bit integers and their ones-complement sum forms the Internet checksum. A receiver calculates the checksum it calculates over the received data and checks whether it matches the checksum carried in the received packet. RFC1071 [RFC 1071] discusses the Internet checksum algorithm and its implementation in detail. In the TCP/IP protocols, the Internet checksum is computed over all fields (header and data fields included). In other protocols, e.g., XTP [Strayer 1992], one checksum is computed over the header, with another checksum computed over the entire packet.

McAuley [McAuley 1994] describe improved weighted checksum codes that are suitable for high-speed software implementation and 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

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.

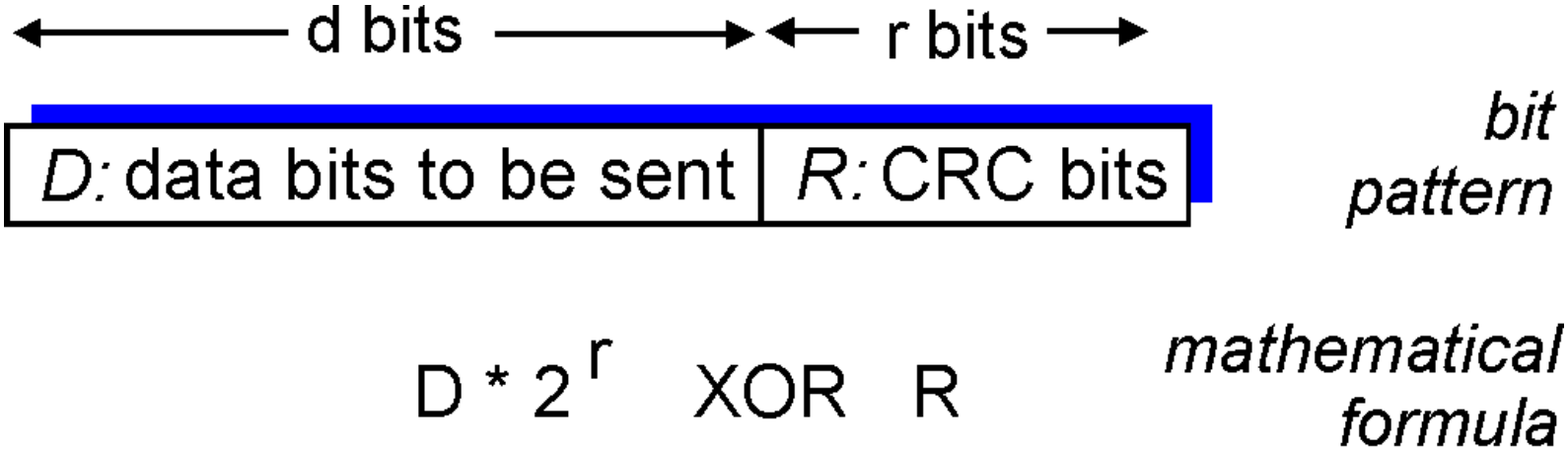

Figure 5.2-4: CRC codes

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 a *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 5.2-4. 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* using modulo 2 arithmetic. The process of error checking with CRC's is thus simple: the receiver divides the *d+r* received bits by *G*. If the remainder is non-zero, 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 binary arithmetic, multiplication by 2^k left shifts a bit pattern by *k* places. Thus, given *D* and *R*, the quantity *D*2r XOR R* yields the *d+r* bit pattern shown in Figure 5.2-4. We'll use this algebraic characterization of the *d+r* bit pattern from Figure 5.2-4 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

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

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

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

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

$$R = \text{remainder}\ (\ D*2r\ /\ G\ )$$

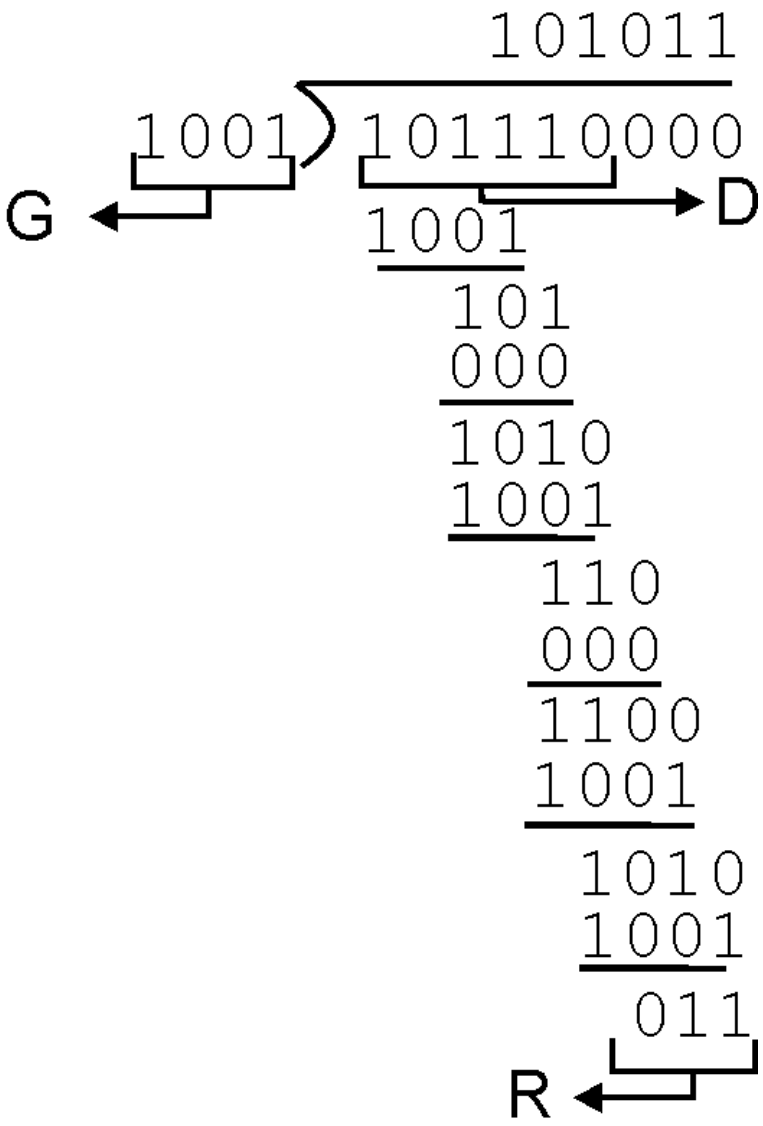

Figure 5.2-5: An example CRC calculation

Figure 5.2-5 illustrates this calculation for the case of D = 101110, d = 6 and G = 1001, r=3. The nine bits transmitted in this case are 101110 011. You should check these calculations for yourself and also check that indeed $D2^r$ = 101011 * G XOR R.

International standards have been defined for 8-, 12-, 16- and 32-bit generators, *G*. An 8-bit CRC is used to protect the 5-byte header in ATM cells. The CRC-32 32-bit standard, which has been adopted in a number of link-level IEEE protocols, uses a generator of

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

Each of the CRC standards can detect burst errors of less than *r+1* bits and any odd number of bit errors. Furthermore, under appropriate assumptions, a burst of length greater than *r+1* bits is detected with probability $1 - 0.5^r$. 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.

References

[Biersak 1992] E.W. Biersack, "Performance evaluation of forward error correction in ATM networks", *Proc. ACM Sigcomm Conference*, (Baltimore, MD 1992), pp. 248-257.
[Byers 1998] J. Byers, M. Luby, M. Mitzenmacher, A Rege, "A digital fountain approach to reliable distribution of bulk data," *Proc. ACM Sigcomm Conference*, (Vancouver, 1998), pp. 56-67
[Feldmeier 1995] D. Feldmeier, "Fast Software Implementation of Error Detection Codes," IEEE/ACM Transactions on Networking, Vol. 3., No. 6 (Dec. 1995), pp. 640 -652.
[Fletcher 1982] J.G. Fletcher, "An Arithmetic Checksum for Serial Transmissions", *IEEE Transactions on Communications*, Vol. 30, No. 1 (January 1982), pp 247-253.
[McAuley 1984] A. McAuley, "Weighted Sum Codes for Error Detection and Their Comparison with Existing Codes", *IEEE/ACM Transactions on Networking*, Vol. 2, No. 1 (February 1994), pp. 16-22.
[Nonnenmacher 1998] J. Nonnenmacher, E. Biersak, D. Towsley, "Parity-Based Loss Recovery for Reliable Multicast Transmission," *IEEE/ACM Transactions on Networking*, Vol. 6, No. 4 (Aug. 1998), Pages 349 - 361.
[RFC 1071] B. Draden, D. Borman, C. Partridge, "Computing the Internet Checksum," RFC 1071, Sept. 1988.
[Rubenstein 1998] D. Rubenstein, J. Kurose, D. Towsley ``Real-Time Reliable Multicast Using Proactive Forward Error Correction" , *Proceedings of NOSSDAV '98* , (Cambridge, UK, July 1998).
[Schwartz 1980] M. Schwartz, *Information, Transmission, Modulation, and Noise*, McGraw Hill, NY, NY 1980.
[Shacham 1990] N. Shacham, P. McKenney, "Packet Recovery in High-Speed Networks Using Coding and Buffer Management", *Proc. IEEE Infocom Conference*, (San Francisco, 1990), pp. 124-131.
[Spragins 1991] J. D. Spragins, *Telecommunications protocols and design* , Addison-Wesley, Reading MA, 1991.
[Strayer 1992] W.T. Strayer, B. Dempsey, A.Weaver, XTP: *The Xpress Transfer Protocol*, Addison Wesley, Reading MA, 1992.
