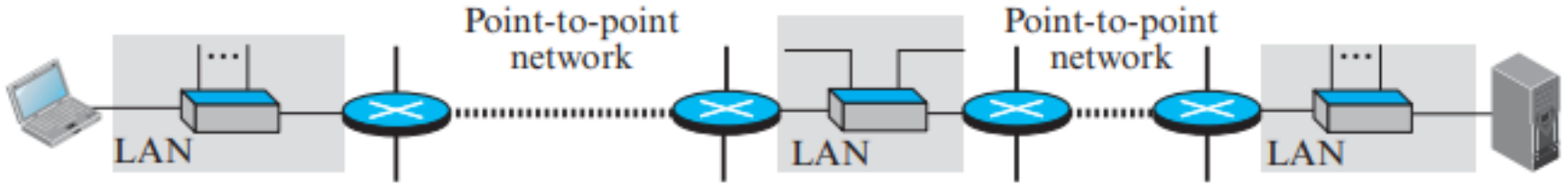
UNIT 2: Data-Link Layer

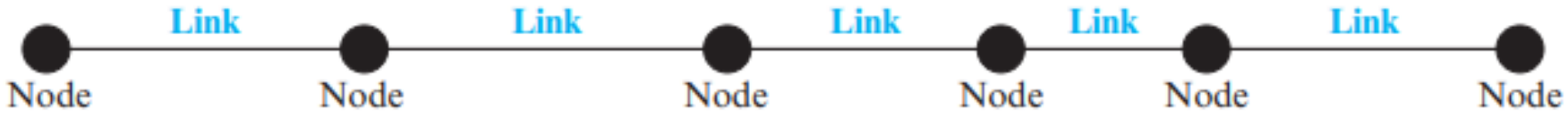
Nodes and Links

communication at the application, transport, and network layers is end-to end, communication at the data-link layer is node-to-node. It is customary to refer to the two end hosts and the routers as nodes and the networks in between as links. The following is a simple representation of links and nodes when the path of the data unit is only six nodes. The first node is the source host; the last node is the destination host. The other four nodes are four routers. The first, the third, and the fifth links represent the three LANs; the second and the fourth links represent the two WANs.

Figure 5.2 *Nodes and Links*



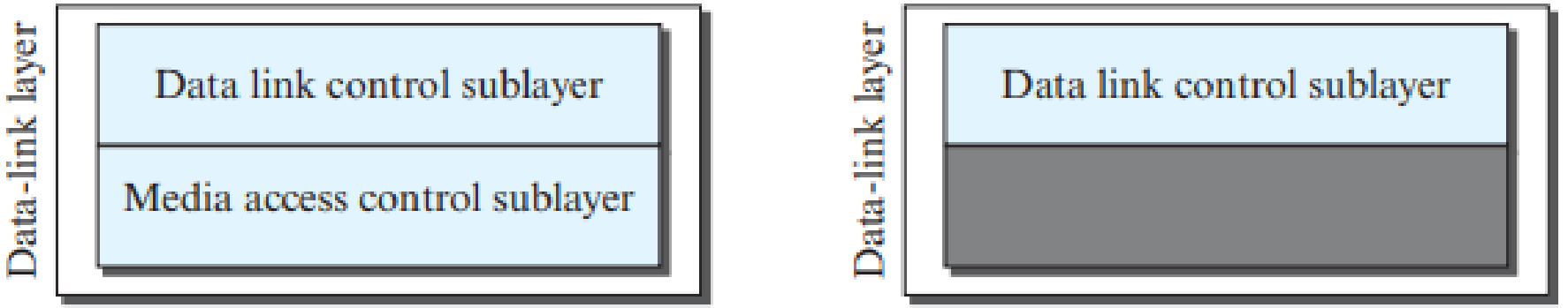
a. A small part of the Internet



b. Nodes and links

Two Sub layers To better understand the functionality of and the services provided by the link layer, we can divide the data-link layer into two sub layers: data link control (DLC) and media access control (MAC). The data link control sub layer deals with all issues common to both point-to-point and broadcast links; the media access control sub layer deals only with issues specific to broadcast links. In other words, we separate theses two types of links at the data-link layer as shown in Figure 5.3

Figure 5.3 *Dividing the data-link layer into two sublayers*



a. Data-link layer of a broadcast link

b. Data-link layer of a point-to-point link

DATA LINK CONTROL (DLC) The data link control deals with procedures for communication between two adjacent nodes—node-to-node communication—no matter whether the link is dedicated or broadcast. Data link control (DLC) functions include framing, flow and error control, and error detection and correction. In this section, we first discuss framing, or how to organize the bits that are carried by the physical layer.

Framing: Data transmission in the physical layer means moving bits in the form of a signal from the source to the destination. The physical layer provides bit synchronization to ensure that the sender and receiver use the same bit durations and timing.

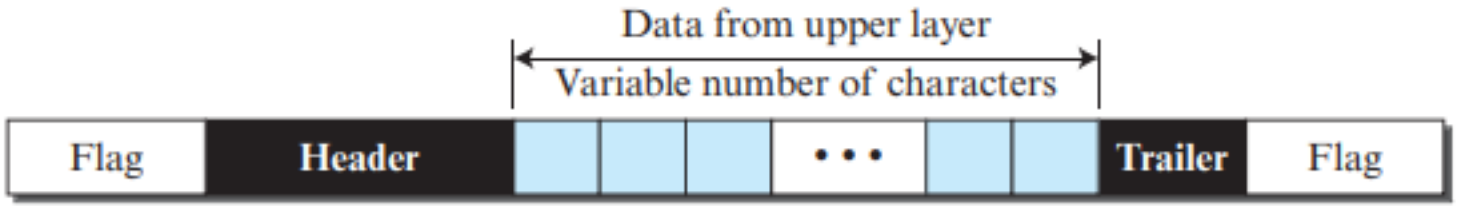
The data-link layer, on the other hand, needs to pack bits into frames, so that each frame is distinguishable from another. Our postal system practices a type of framing. The simple act of inserting a letter into an envelope separates one piece of information from another; the envelope serves as the delimiter. In addition, each envelope defines the sender and receiver addresses, which is necessary since the postal system is a many to-many carrier facility. Figure 5.3 Dividing the data-link layer into two sub layers Data-link layer Data link control sub layer Media access control sub layer Data-link layer Data link control sub layer a. Data-link layer of a broadcast link b. Data-link layer of a point-to-point link SECTION
Framing in the data-link layer separates a message from one source to a destination by adding a sender address and a destination address. The destination address defines where the packet is to go; the sender address helps the recipient acknowledge the receipt. Although the whole message could be packed in one frame, that is not normally done. One reason is that a frame can be very large, making flow and error control very inefficient. When a message is carried in one very large frame, even a single-bit error would require the retransmission of the whole frame. When a message is divided into smaller frames, a single-bit error affects only that small frame.

Frame Size Frames can be of fixed or variable size. In fixed-size framing, there is no need for defining the boundaries of the frames

character-oriented approach and a bit-oriented approach.

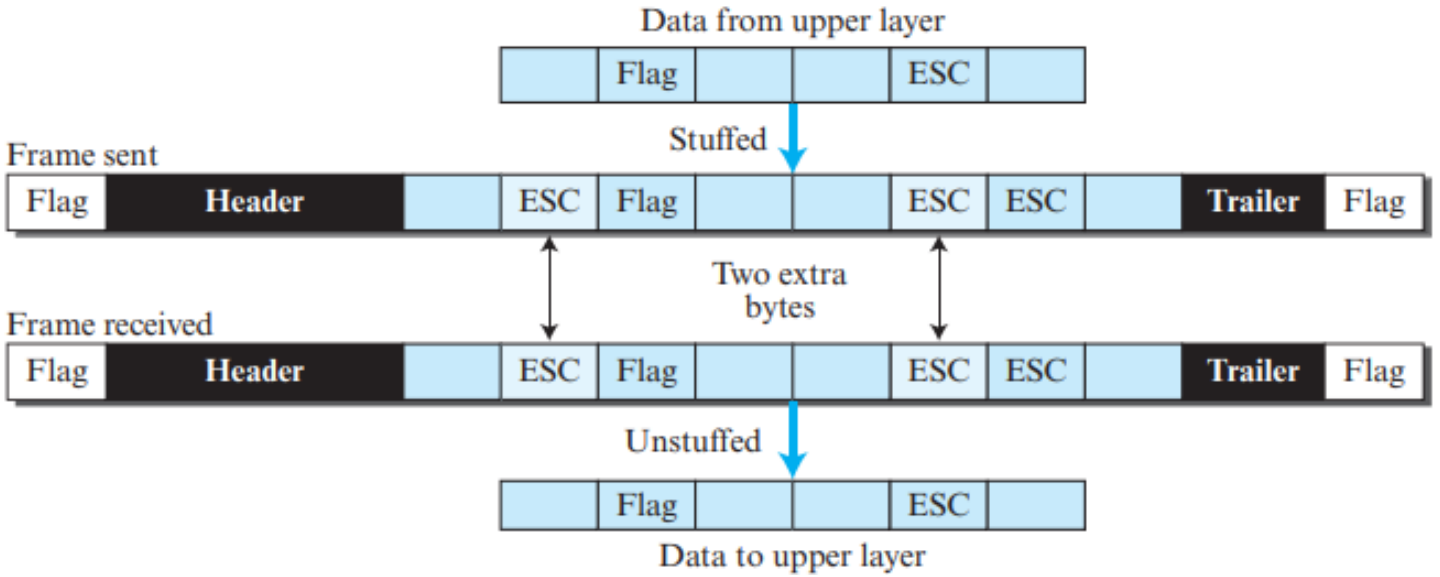
Character-Oriented Framing In character-oriented (or byte-oriented) framing, data to be carried are 8-bit characters from a coding system such as ASCII (see Appendix A). The header, which normally carries the source and destination addresses and other control information, and the trailer, which carries error detection redundant bits, are also multiples of 8 bits. To separate one frame from the next, an 8-bit (1-byte) flag is added at the beginning and the end of a frame. The flag, composed of protocol-dependent special characters, signals the start or end of a frame. Figure 5.4 shows the format of a frame in a character-oriented protocol.

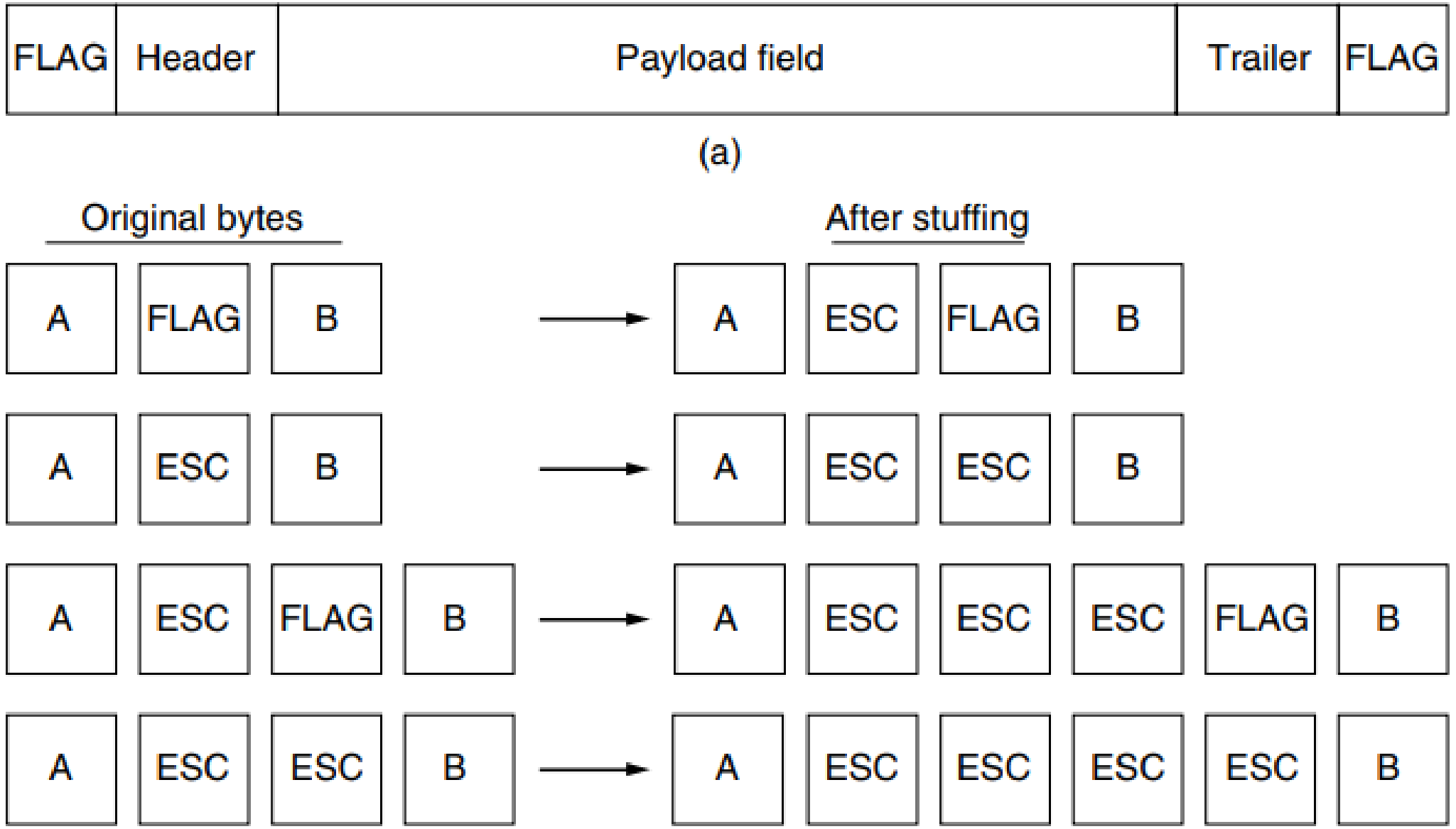
Figure 5.4 *A frame in a character-oriented protocol*



Character-oriented framing was popular when only text was exchanged by the data-link layers. The flag could be selected to be any character not used for text communication. Now, however, we send other types of information such as graphs, audio, and video, any pattern used for the flag could also be part of the information. If this happens, the receiver, when it encounters this pattern in the middle of the data, thinks it has reached the end of the frame. To fix this problem, a byte-stuffing strategy was added to character-oriented framing. In byte stuffing (or character stuffing), a special byte is added to the data section of the frame when there is a character with the same pattern as the flag. The data section is stuffed with an extra byte. This byte is usually called the escape character (ESC) and has a predefined bit pattern. Whenever the receiver encounters the ESC character, it removes it from the data section and treats the next character as data, not as a delimiting flag. Byte stuffing by the escape character allows the presence of the flag in the data section of the frame, but it creates another problem. What happens if the text contains one or more escape characters followed by a byte with the same pattern as the flag? The receiver removes the escape character, but keeps the next byte, which is incorrectly interpreted as the end of the frame. To solve this problem, the escape characters that are part of the text must also be marked by another escape character. In other words, if the escape character is part of the text, an extra one is added to show that the second one is part of the text.

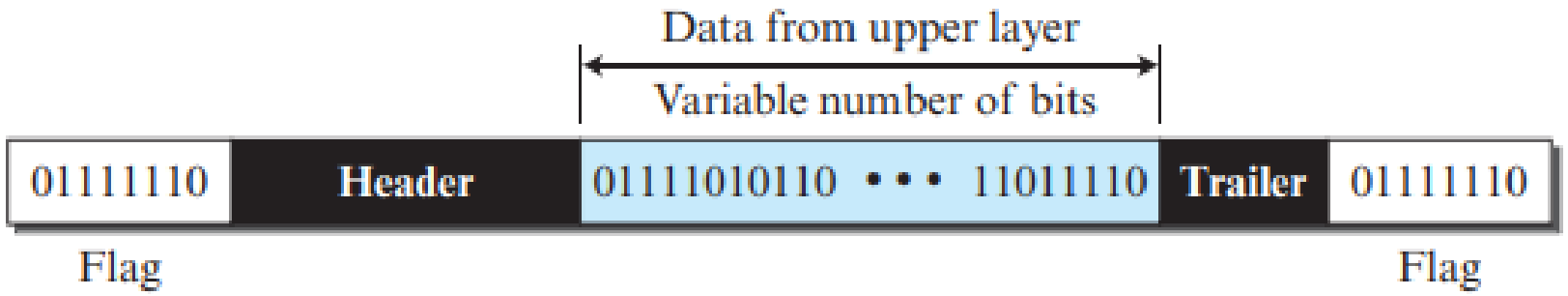
Figure 5.5 *Byte stuffing and unstuffing*




FLAG
Header
Payload field
Trailer
FLAG
(a)
Original bytes
After stuffing
A
FLAG
B
A
ESC
FLAG
B
A
ESC
B
A
ESC
ESC
B
A
ESC
FLAG
B
A
ESC
ESC
ESC
FLAG
B
A
ESC
ESC
B
A
ESC
ESC
ESC
ESC
B

Bit-Oriented Framing In bit-oriented framing, the data section of a frame is a sequence of bits to be interpreted by the upper layer as text, graphic, audio, video, and so on. However, in addition to headers (and possible trailers), we still need a delimiter to separate one frame from the other. Most protocols use a special 8-bit pattern flag, 01111110, as the delimiter to define the beginning and the end of the frame, as shown in Figure 5.6. This flag can create the same type of problem we saw in the character-oriented protocols. That is, if the flag pattern appears in the data, we need to somehow inform the receiver that this is not the end of the frame. We do this by stuffing 1 single bit (instead of 1 byte) to prevent the pattern from looking like a flag. The strategy is called bit stuffing. In bit stuffing, if a 0 and five consecutive 1 bits are encountered, an extra 0 is added. This extra stuffed bit is eventually removed from the data by the receiver.

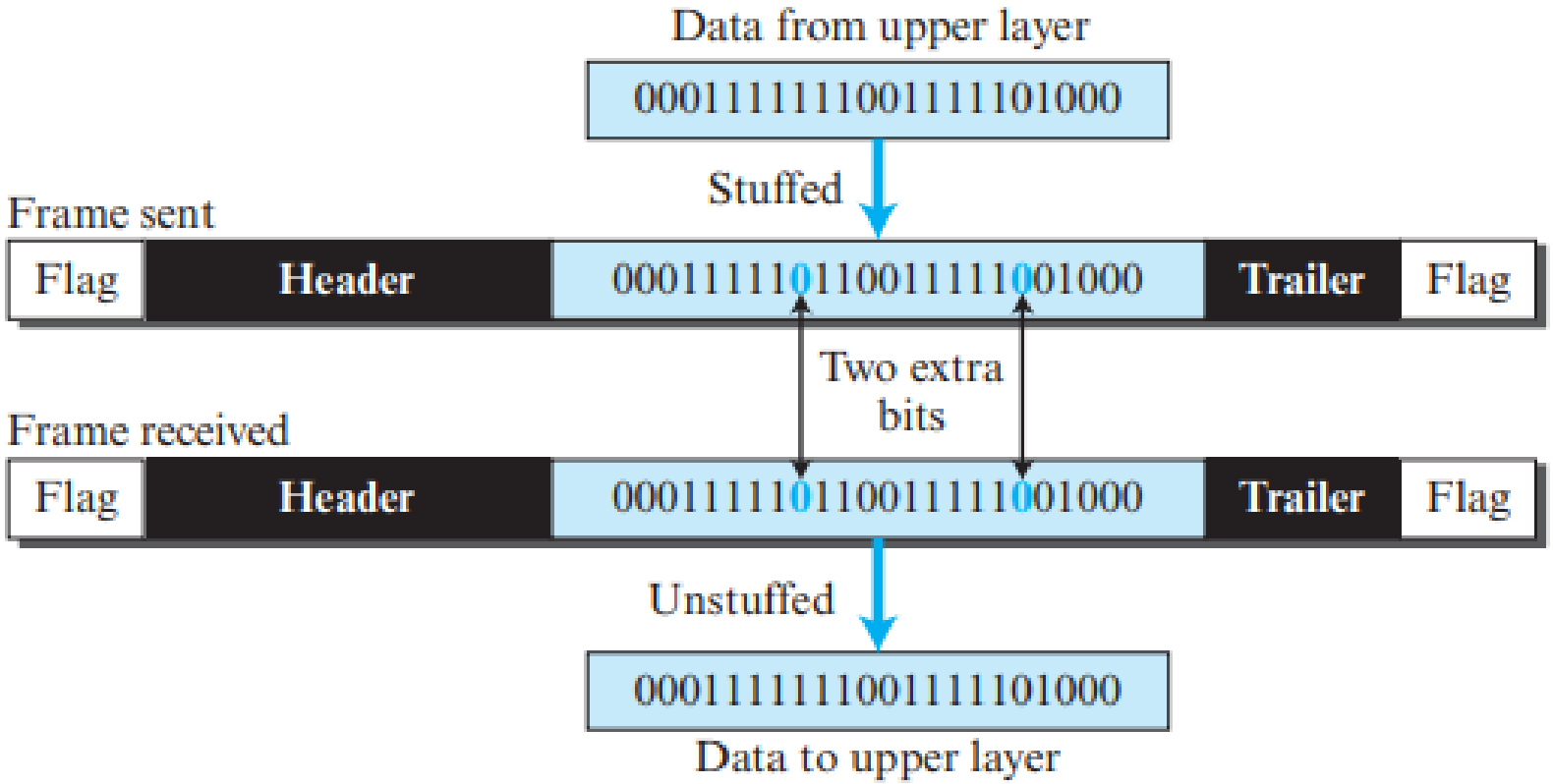
Figure 5.6 *A frame in a bit-oriented protocol*



Note that the extra bit is added after one 0 followed by five 1s regardless of the value of the next bit. This guarantees that the flag field sequence does not inadvertently appear in the frame.

Figure 5.7 shows bit stuffing at the sender and bit removal at the receiver. Note that even if we have a 0 after five 1s, we still stuff a 0. The 0 will be removed by the receiver.

Figure 5.7 *Bit stuffing and unstuffing*



This means that if the flag like pattern 01111110 appears in the data, it will change to 011111010 (stuffed) and is not mistaken for a flag by the receiver. The real flag 01111110 is not stuffed by the sender and is recognized by the receiver.

Error Control

Error control is both error detection and error correction. It allows the receiver to inform the sender of any frames lost or damaged in transmission and coordinates the retransmission of those frames by the sender. In the data-link layer, the term error control refers primarily to methods of error detection and retransmission. Error control in the data-link layer is often implemented simply: Any time an error is detected in an exchange, corrupted frames are retransmitted. We need, however, to emphasize that the error control at the transport layer is end-to-end, but the error control at the data-link layer is node-to-node, across the link. In other words, each time a frame passes through a link, we need to be sure that it is not corrupted.

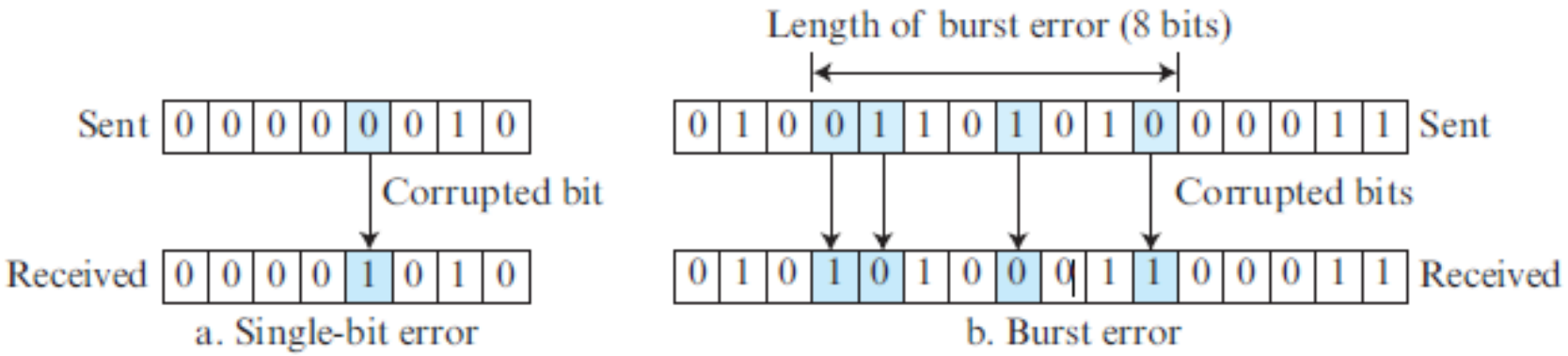
5.2.3 Error Detection and Correction

At the data-link layer, if a frame is corrupted between the two nodes, it needs to be corrected before it continues its journey to other nodes. However, most link-layer protocols simply discard the frame and let the upper-layer protocols handle the retransmission of the frame. Some wireless protocols, however, try to correct the corrupted frame.

Types of Errors

Whenever bits flow from one point to another, they are subject to unpredictable changes because of interference. This interference can change the shape of the signal. The term single-bit error means that only 1 bit of a given data unit (such as a byte, character, or packet) is changed from 1 to 0 or from 0 to 1. The term burst error means that 2 or more bits in the data unit have changed from 1 to 0 or from 0 to 1. Figure 5.8 shows the effect of a single-bit and burst error on a data unit.

Figure 5.8 *Single-bit and burst error*



A burst error is more likely to occur than a single-bit error because the duration of the noise signal is normally longer than the duration of 1 bit, which means that when noise affects data, it affects a set of bits. The number of bits affected depends on the data rate and duration of noise. For example, if we are sending data at 1 kbps, a noise of 1/100 seconds can affect 10 bits; if we are sending data at 1 Mbps, the same noise can affect 10,000 bits.

Redundancy:

The central concept in detecting or correcting errors is redundancy. To be able to detect or correct errors, we need to send some extra bits with our data. These redundant bits are added by the sender and removed by the receiver. Their presence allows the receiver to detect or correct corrupted bits.

Detection versus Correction

The correction of errors is more difficult than the detection. In error detection, we are looking only to see if any error has occurred. The answer is a simple yes or no. We are not even interested in the number of corrupted bits. A single-bit error is the same for us as a burst error. In error correction, we need to know the exact number of bits that are corrupted and, more importantly, their location in the message. The number of the errors and the size of the message are important factors. If we need to correct a single error in an 8-bit data unit, we need to consider eight possible error locations; if we need to correct two errors in a data unit of the same size, we need to consider 28 (permutation of 8 by 2) possibilities. You can imagine the receiver's difficulty in finding 10 errors in a data unit of 1000 bits

Coding

Redundancy is achieved through various coding schemes. The sender adds redundant bits through a process that creates a relationship between the redundant bits and the actual data bits. The receiver checks the relationships between the two sets of bits to detect errors. The ratio of redundant bits to the data bits and the robustness of the process are important factors in any coding scheme**. We can divide coding schemes into two broad categories: block coding and convolution coding.**

Block Coding

In block coding, we divide our message into blocks, **each of k bits, called datawords**.
We add r redundant bits to each block to make the length n = k + r.
The resulting n-bit blocks are called **codewords**.
it is important to know that we have a set of datawords, each of size k, and a set of codewords, each of size of n.
With k bits, we can create a combination of 2^k datawords;
with n bits, we can create a combination of 2^n codewords.
Since n > k, the number of possible codewords is larger than the number of possible datawords. The block coding process is one-to-one; the same dataword is always encoded as the same codeword.
This means that we have $2^n - 2^k$ codewords that are not used.
We call these codewords invalid or illegal.
The trick in error detection is the existence of these invalid codes, as we discuss next. If the receiver receives an invalid codeword, this indicates that the data was corrupted during transmission.

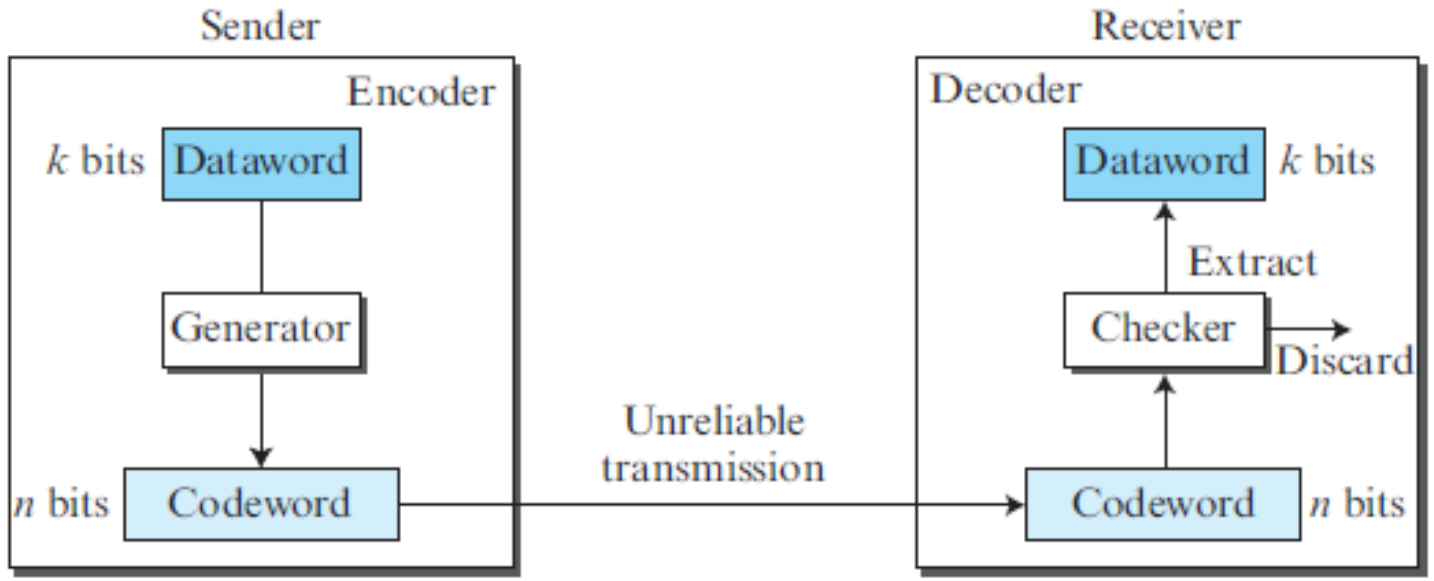
Error Detection

How can errors be detected by using block coding? If the following two conditions are met, the receiver can detect a change in the original codeword.

1. The receiver has (or can find) a list of valid codewords.
2. The original codeword has changed to an invalid one.

Figure 5.9 shows the role of block coding in error detection

Figure 5.9 *Process of error detection in block coding*



The sender creates codewords out of datawords by using a generator that applies the rules and procedures of encoding. Each codeword sent to the receiver may change during transmission. If the received codeword is the same as one of the valid codewords, the word is accepted; the corresponding dataword is extracted for use. If the received codeword is not valid, it is discarded. However, if the codeword is corrupted during transmission but the received word still matches a valid codeword, the error remains undetected.

Example 5.1

Let us assume that k = 2 and n = 3. Table 5.1 shows the list of datawords and codewords. Later, we will see how to derive a codeword from a dataword.

Assume the sender encodes the dataword 01 as 011 and sends it to the receiver. Consider the following cases:

1. The receiver receives 011. It is a valid codeword. The receiver extracts the dataword 01 from it.
2. The codeword is corrupted during transmission, and 111 is received (the leftmost bit is corrupted). This is not a valid codeword and is discarded.
3. The codeword is corrupted during transmission, and 000 is received (the right two bits are corrupted). This is a valid codeword. The receiver incorrectly extracts the dataword 00. Two corrupted bits have made the error undetectable.

Table 5.1 *A code for error detection in Example 5.1*

Datawords	*Codewords*	*Datawords*	*Codewords*
00	000	10	101
01	011	11	110

Hamming Distance

One of the central concepts in coding for error control is the idea of the Hamming distance. The Hamming distance between two words (of the same size) is the number of differences between the corresponding bits. We show the Hamming distance between two words x and y as d(x, y). We may wonder why Hamming distance is important for error detection. **The reason is that the Hamming distance between the received codeword and the sent codeword is the number of bits that are corrupted during transmission**.

For example,

if the codeword 00000 is sent and 01101 is received, 3 bits are in error and the Hamming distance between the two is d(00000, 01101) = 3.

In other words, if the Hamming distance between the sent and the received codeword is not zero, the codeword has been corrupted during transmission.

The Hamming distance can easily be found if we apply the XOR operation (⊕) on the two words and count the number of 1s in the result.

Example 5.2

Let us find the Hamming distance between two pairs of words.

1. The Hamming distance d (000, 011) is 2 because (000 $\oplus$ 011) is 011 (two 1s).
2. The Hamming distance d (10101, 11110) is 3 because (10101 $\oplus$ 11110) is 01011 (three 1s).

Parity-Check Code

Perhaps the most familiar error-detecting code is the parity-check code. This code is a linear block code. In this code, a k-bit dataword is changed to an n-bit codeword where

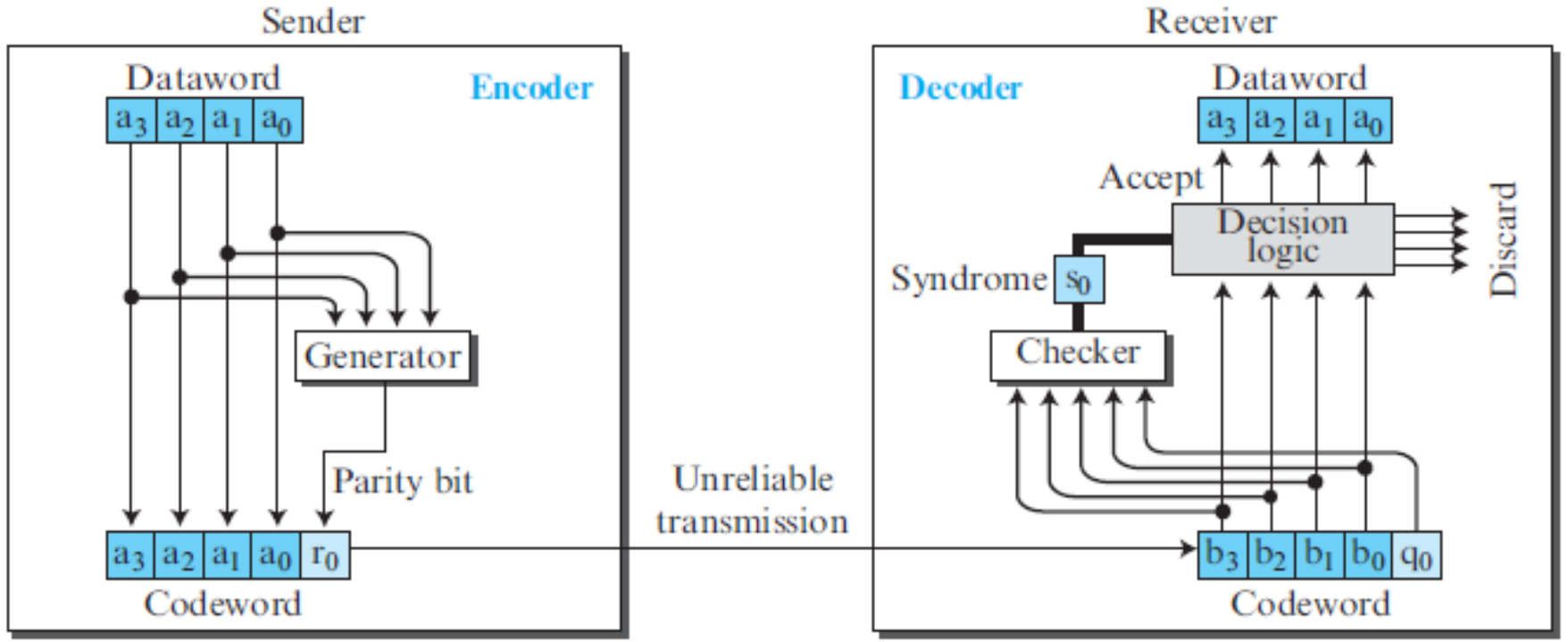
n = k + 1. The extra bit, called the parity bit, is selected to make the total number of 1s in the codeword even. Although some implementations specify an odd number of 1s, we discuss the even case. The minimum Hamming distance for this category is $d_{min} = 2$, which means that the code is a single-bit error-detecting code. Our first code (Table 5.1) is a parity-check code (k = 2 and n = 3). The code in Table 5.2 is also a parity-check code with k = 4 and n = 5.

Table 5.2 *Simple parity-check code C(5, 4)*

Datawords	Codewords	*Datawords*	Codewords
0000	00000	1000	10001
0001	00011	1001	10010
0010	00101	1010	10100
0011	00110	1011	10111
0100	01001	1100	11000
0101	01010	1101	11011
0110	01100	1110	11101
0111	01111	1111	11110

Figure 5.11 shows a possible structure of an encoder (at the sender) and a decoder (at the receiver).

Figure 5.11 *Encoder and decoder for simple parity-check code*



Example 5.7

Let us look at some transmission scenarios. Assume the sender sends the dataword 1011. The codeword created from this dataword is 10111, which is sent to the receiver.

We examine five cases:

1. No error occurs; the received codeword is 10111. The syndrome is 0. The dataword 1011 is created.
2. One single-bit error changes a1. The received codeword is 10011. The syndrome is 1. No dataword is created.
3. One single-bit error changes r0. The received codeword is 10110. The syndrome is 1. No dataword is created. Note that although none of the dataword bits are corrupted, no dataword is created because the code is not sophisticated enough to show the position of the corrupted bit.
4. An error changes r0 and a second error changes a3. The received codeword is 00110. The syndrome is 0. The dataword 0011 is created at the receiver. Note that here the dataword is wrongly created due to the syndrome value. The simple parity-check decoder cannot detect an even number of errors. The errors cancel each other out and give the syndrome a value of 0.
5. Three bits—a3, a2, and a1—are changed by errors. The received codeword is 01011. The syndrome is 1. The dataword is not created. This shows that the simple parity check, guaranteed to detect one single error, can also find any odd number of errors.

Cyclic Codes Cyclic codes are special linear block codes with one extra property. In a cyclic code, if a codeword is cyclically shifted (rotated), the result is another codeword.

For example, if 1011000 is a codeword and we cyclically left-shift, then 0110001 is also a codeword. In this case, if we call the bits in the first word a0 to a6, and the bits in the second word b0 to b6, we can shift the bits by using the following: In the rightmost equation, the last bit of the first word is wrapped around and becomes the first bit of the second word.

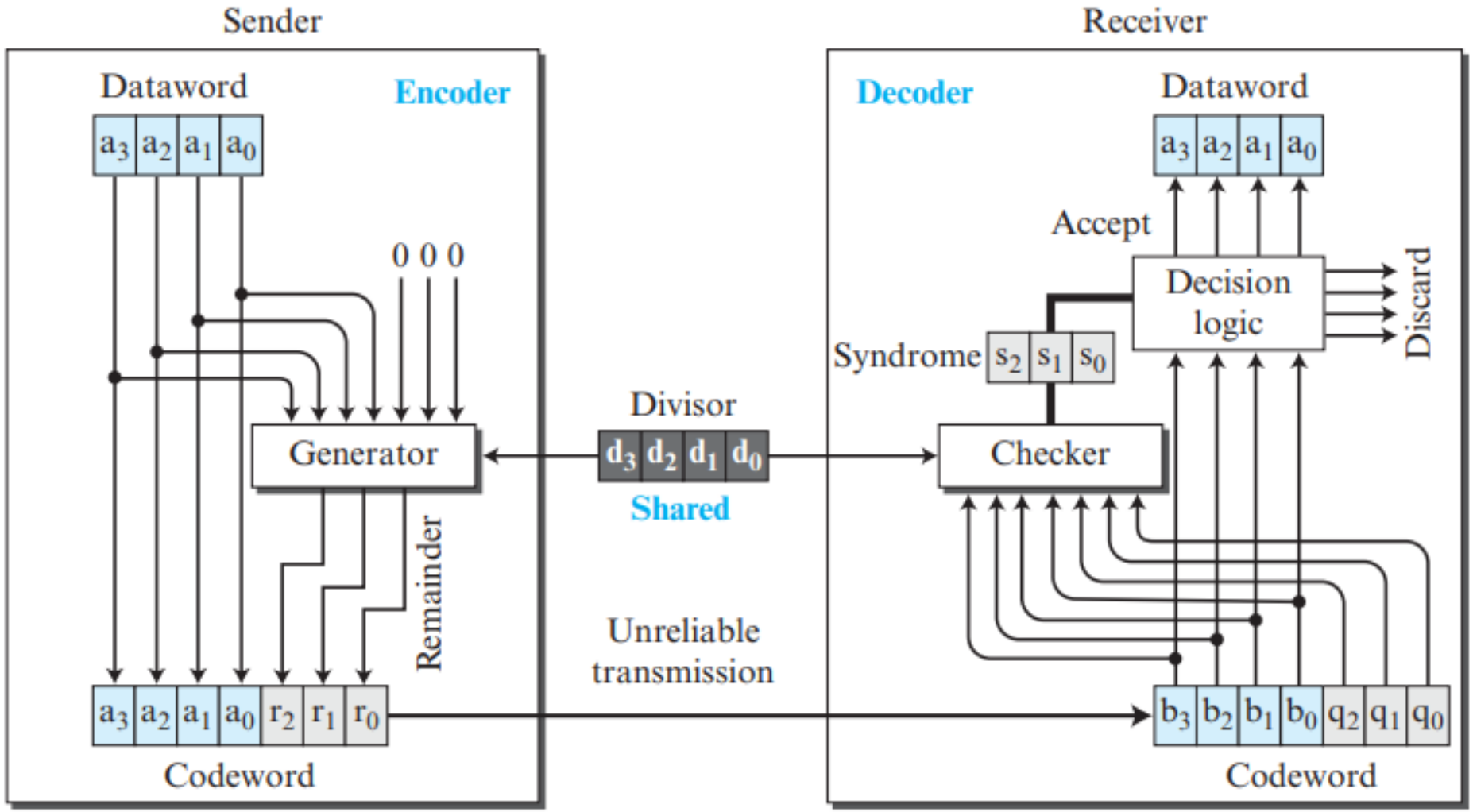
Cyclic Redundancy Check We can create cyclic codes to correct errors.

cyclic codes called the cyclic redundancy check (CRC) that is used in networks such as LANs and WANs. Table 5.3 shows an example of a CRC code. We can see both the linear and cyclic properties of this code.

Table 5.3 *A CRC code with C(7, 4)*

Dataword	*Codeword*	*Dataword*	*Codeword*
0000	0000000	1000	1000101
0001	0001011	1001	1001110
0010	0010110	1010	1010011
0011	0011101	1011	1011000
0100	0100111	1100	1100010
0101	0101100	1101	1101001
0110	0110001	1110	1110100
0111	0111010	1111	1111111

Figure 5.12 *CRC encoder and decoder*



In the encoder, the dataword has k bits (4 here); the codeword has n bits (7 here). The size of the dataword is augmented by adding n − k (3 here) 0s to the right-hand side of the word. The n-bit result is fed into the generator. The generator uses a divisor of size n − k + 1 (4 here), predefined and agreed upon. The generator divides the augmented dataword by the divisor (modulo-2 division). The quotient of the division is discarded; the remainder (r2r1r0) is appended to the dataword to create the codeword. The decoder receives the codeword (possibly corrupted in transition). A copy of all n bits is fed to the checker, which is a replica of the generator. The remainder produced by the checker is a syndrome of n − k (3 here) bits, which is fed to the decision logic analyzer. The analyzer has a simple function. If the syndrome bits are all 0s, the 4 leftmost bits of the codeword are accepted as the dataword (interpreted as no error); otherwise, the 4 bits are discarded (error). Encoder Let us take a closer look at the encoder. The encoder takes a dataword and augments it with n − k number of 0s. It then divides the augmented dataword by the divisor, as shown in Figure 5.13.

Figure 5.13 *Division in CRC encoder*

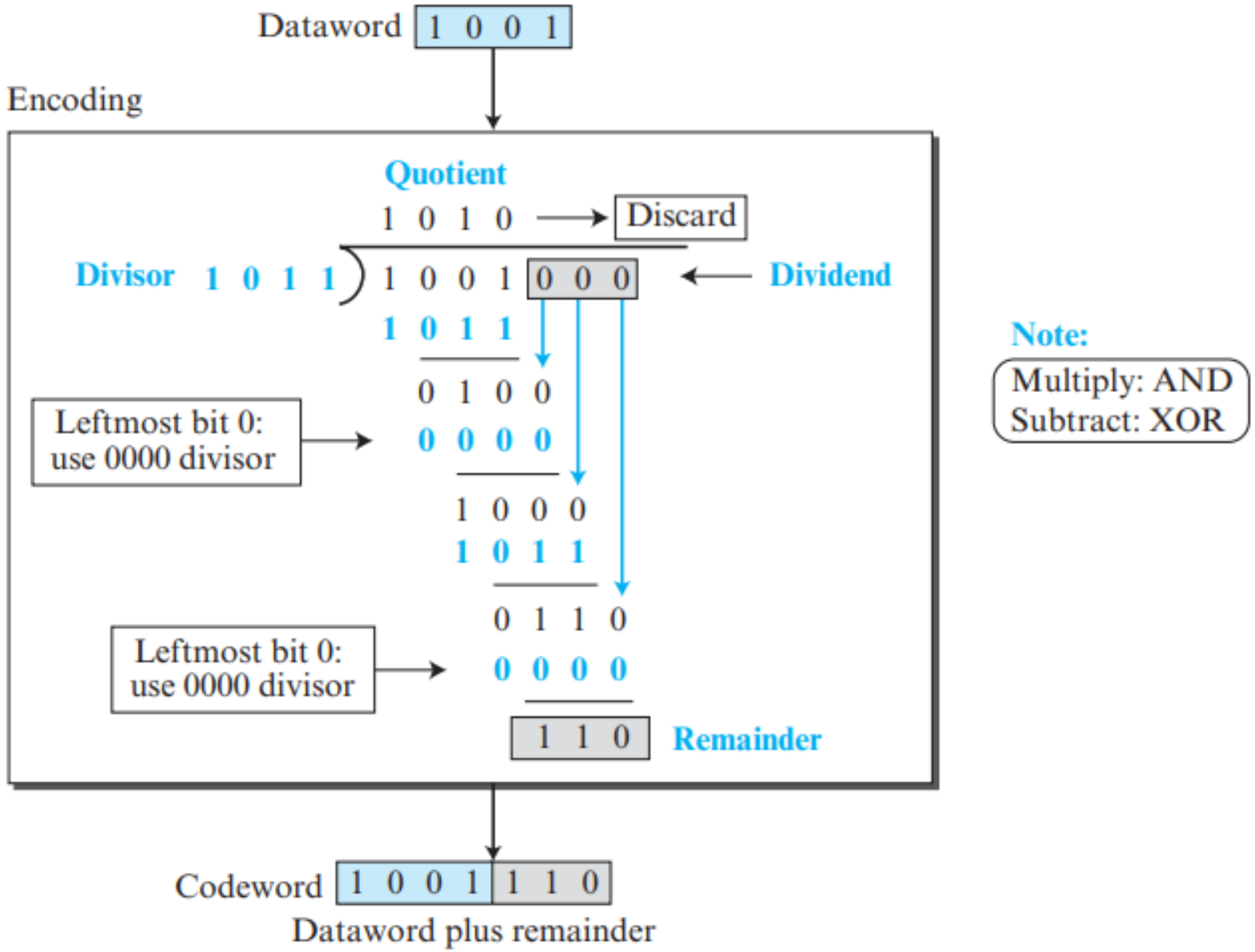
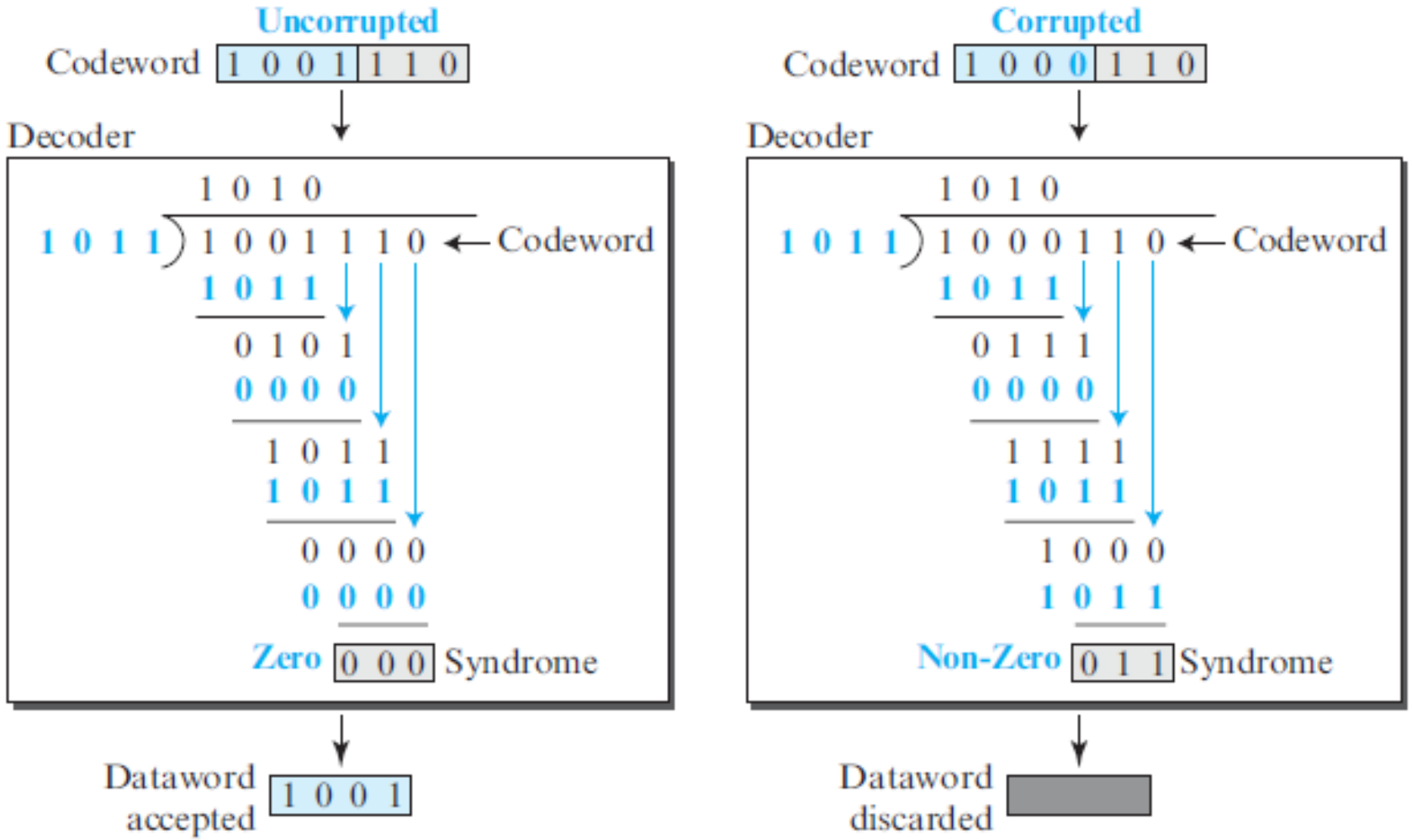


Figure 5.14 *Division in the CRC decoder for two cases*



The process of modulo-2 binary division is the same as the familiar division process we use for decimal numbers. However, addition and subtraction in this case are the same; we use the XOR operation to do both. As in decimal division, the process is done step by step. In each step, a copy of the divisor is XORed with the 4 bits of the dividend. The result of the XOR operation (remainder) is 3 bits (in this case), which is used for the next step after 1 extra bit is pulled down to make it 4 bits long. There is one important point we need to remember in this type of division. If the leftmost bit of the dividend (or the part used in each step) is 0, the step cannot use the regular divisor; we need to use an all-0s divisor. When there are no bits left to pull down, we have a result. The 3-bit remainder forms the check bits (r2, r1, and r0). They are appended to the dataword to create the codeword.

Decoder The codeword can change during transmission. The decoder does the same division process as the encoder. The remainder of the division is the syndrome. If the syndrome is all 0s, there is no error with a high probability; the dataword is separated from the received codeword and accepted. Otherwise, everything is discarded.

Figure 5.14 shows two cases: The left-hand figure shows the value of the syndrome when no error has occurred; the syndrome is 000. The right-hand part of the figure shows the case in which there is a single error. The syndrome is not all 0s (it is 011).

Divisor We may be wondering how the divisor 1011 is chosen. This depends on the expectation we have from the code. We discuss the criteria on the book website. Some of the standard divisors used in networking are shown in Table 5.4. The number in the name of the divisor (for example, CRC-32) refers to the degree of polynomial (the highest power) representing the divisor. The number of bits is always one more than the degree of polynomial. For example, CRC-8 has 9 bits and CRC-32 has 33 bits.

Table 5.4 *Standard polynomials*

Name	*Binary*
CRC-8	100000111
CRC-10	11000110101
CRC-16	1000100000100001
CRC-32	100000100110000010001110110110111

5.2.4 Two DLC Protocols After finishing all issues related to the DLC sublayer, we discuss two DLC protocols that actually implemented those concepts. The first, **HDLC, is the base of many protocols that have been designed for LANs. The second, Point-to-Point, is a protocol derived from HDLC and is used for point-to-point links.**

HDLC High-level Data Link Control (HDLC) is a bit-oriented protocol for communication over point-to-point and multipoint links.

Configurations and Transfer Modes HDLC provides two common transfer modes that can be used in different configurations: **normal response mode (NRM) and asynchronous balanced mode (ABM).**
In normal response mode (NRM), the station configuration is unbalanced. We have one primary station and multiple secondary stations. A primary station can send commands; a secondary station can only respond. The NRM is used for both point-to-point and multipoint links, as shown in Figure 5.20.

Figure 5.20 *Normal response mode*

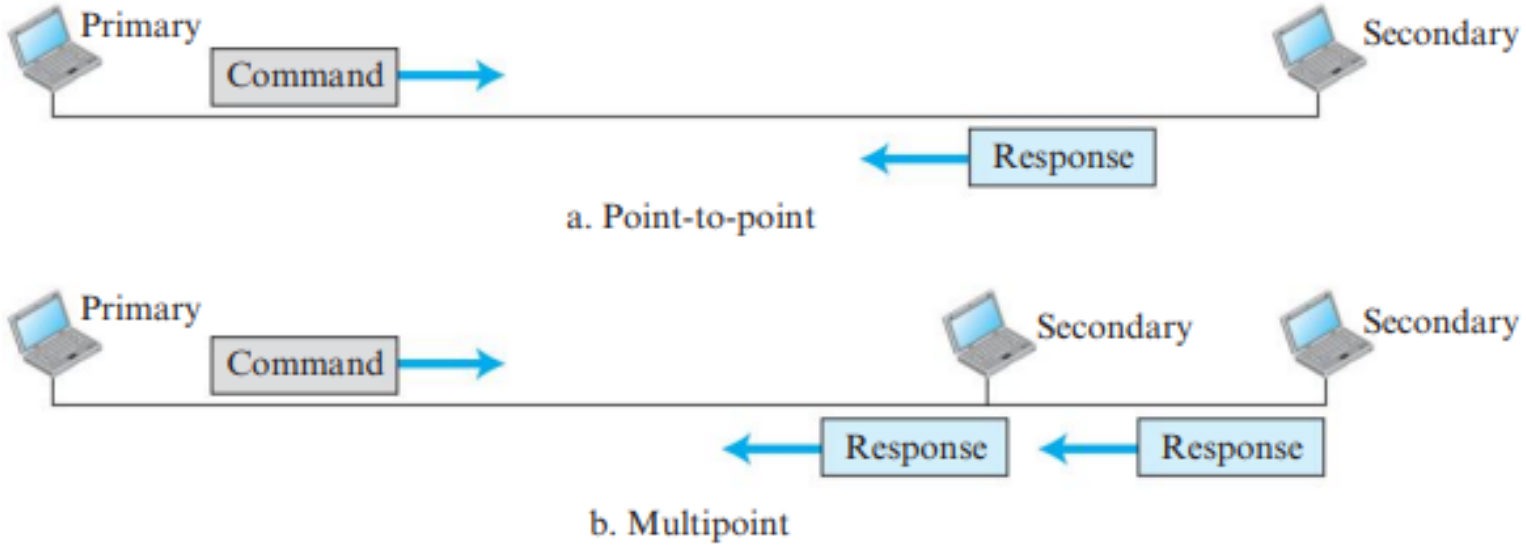
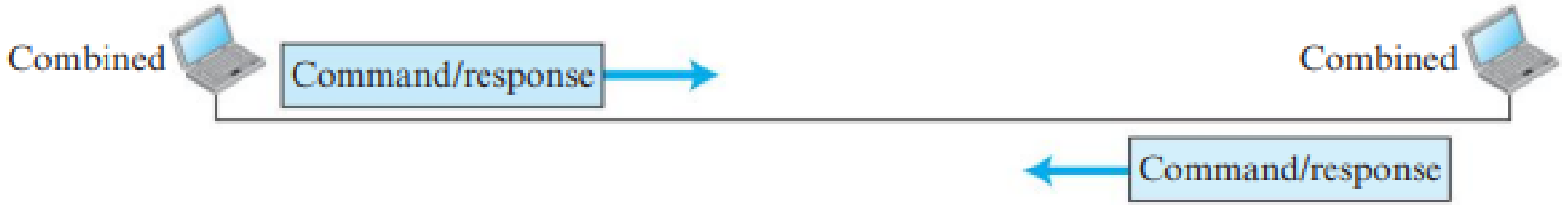
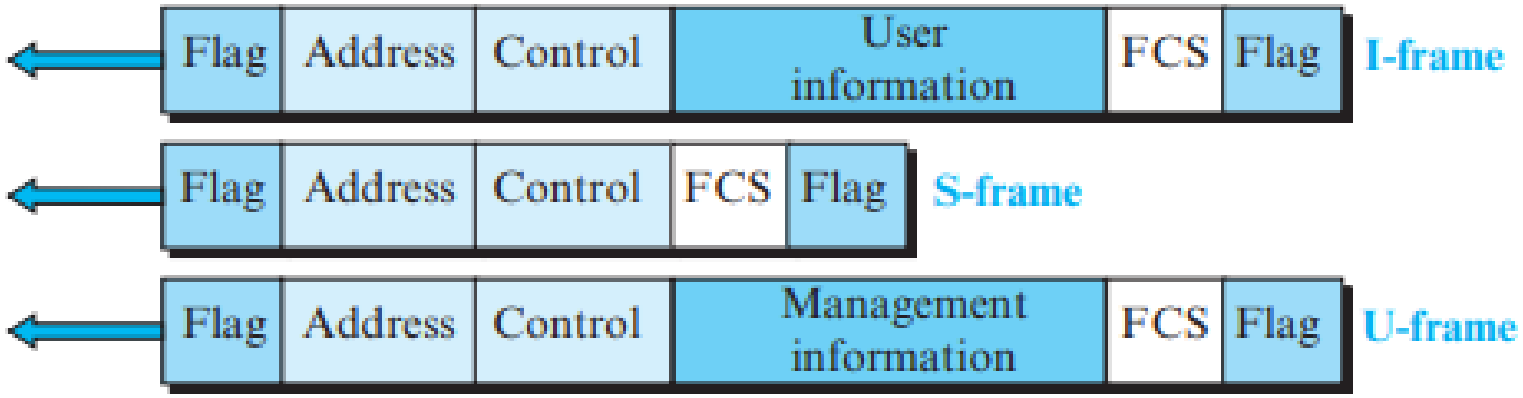


Figure 5.21 *Asynchronous balanced mode*



In ABM, the configuration is balanced. The link is point-to-point, and each station can function as a primary and a secondary (acting as peers), as shown in Figure 5.21. This is the common mode today.

Figure 5.22 *HDLC frames*



Frames To provide the flexibility necessary to support all the options possible in the modes and configurations just described, HDLC defines three types of frames: information frames (I-frames), supervisory frames (S-frames), and unnumbered frames (U-frames). Each type of frame serves as an envelope for the transmission of a different type of message. I frames are used to transport user data and control information relating to user data (piggybacking). S-frames are used only to transport control information. U-frames are reserved for system management. Information carried by U-frames is intended for managing the link itself. Each frame in HDLC may contain up to six fields, as shown in Figure 5.22: a beginning flag field, an address field, a control field, an information field, a frame check sequence (FCS) field, and an ending flag field. In multiple-frame transmissions, the ending flag of one frame can serve as the beginning flag of the next frame.

Let us now discuss the fields and their use in different frame types.

Flag field: This field contains synchronization pattern 01111110, which identifies both the beginning and the end of a frame.

Address field: This field contains the address of the secondary station. If a primary station created the frame, it contains a to address. If a secondary station creates the frame, it contains a from address. The address field can be one byte or several bytes long, depending on the needs of the network.

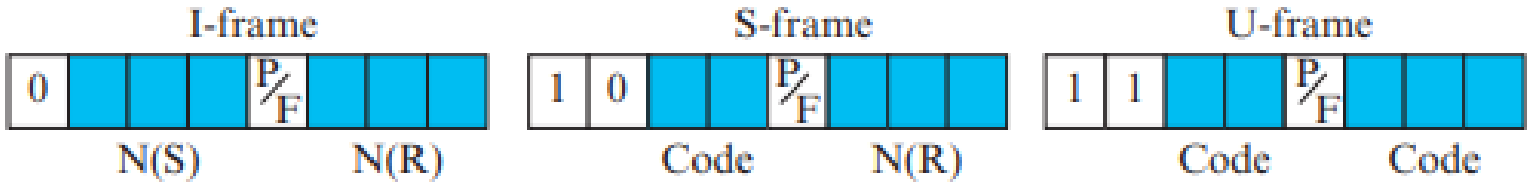
Control field: The control field is one or two bytes used for flow and error control. The interpretation of bits are discussed later.

Information field: The information field contains the user's data from the network layer or management information. Its length can vary from one network to another.

FCS field: The frame check sequence (FCS) is the HDLC error detection field. It can contain either a 2- or 4-byte CRC.

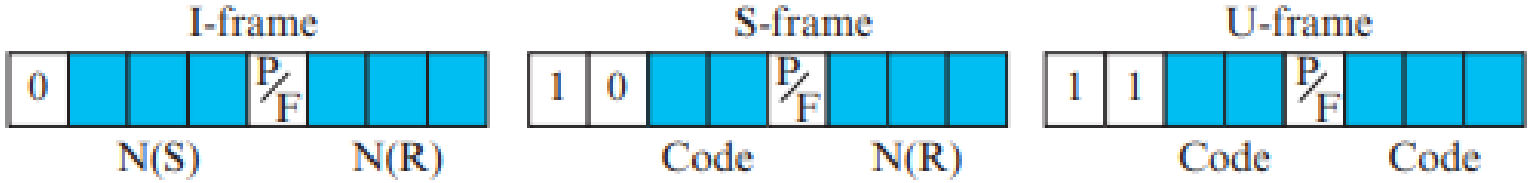
The control field determines the type of frame and defines its functionality. So let us discuss the format of this field in detail. The format is specific for the type of frame, as shown in Figure 5.23.

Figure 5.23 *Control field format for the different frame types*



Control Field for I-Frames: I-frames are designed to carry user data from the network layer. In addition, they can include flow- and error-control information (piggybacking). The subfields in the control field are used to define these functions. The first bit defines the type. If the first bit of the control field is 0, this means the frame is an I-frame. The next 3 bits, called N(S), define the sequence number of the frame. Note that with 3 bits, we can define a sequence number between 0 and 7. The last 3 bits, called N(R), correspond to the acknowledgment number when piggybacking is used. The single bit between N(S) and N(R) is called the P/F bit. The P/F field is a single bit with a dual purpose. It has meaning only when it is set (bit = 1) and can mean poll or final. It means poll when the frame is sent by a primary station to a secondary (when the address field contains the address of the receiver). It means final when the frame is sent by a secondary to a primary (when the address field contains the address of the sender).

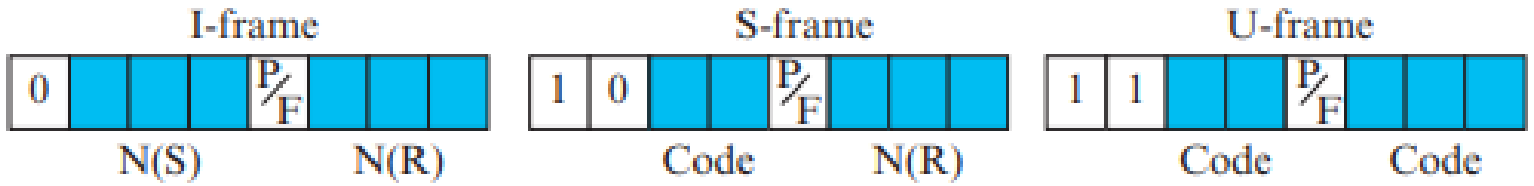
Figure 5.23 *Control field format for the different frame types*



Control Field for S-Frames Supervisory frames are used for flow and error control whenever piggybacking is either impossible or inappropriate. S-frames do not have information fields. If the first 2 bits of the control field are 10, this means the frame is an Sframe. The last 3 bits, called N(R), corresponds to the acknowledgment number (ACK) or negative acknowledgment number (NAK) depending on the type of S-frame. The 2 bits called code are used to define the type of S-frame itself. With 2 bits, we can have four types of S-frames, as described below:

Receive ready (RR). If the value of the code subfield is 00, it is an RR S-frame. This kind of frame acknowledges the receipt of a safe and sound frame or group of frames. In this case, the value of the N(R) field defines the acknowledgment number. q
Receive not ready (RNR). If the value of the code subfield is 10, it is an RNR Sframe. This kind of frame is an RR frame with additional functions. It acknowledges the receipt of a frame or group of frames, and it announces that the receiver is busy and cannot receive more frames. It acts as a kind of congestion-control mechanism by asking the sender to slow down. The value of N(R) is the acknowledgment number.
Reject (REJ). If the value of the code subfield is 01, it is an REJ S-frame. This is a NAK frame, but not like the one used for Selective Repeat ARQ. It is a NAK that can be used in Go-Back-N ARQ to improve the efficiency of the process by informing the sender, before the sender timer expires, that the last frame is lost or damaged. The value of N(R) is the negative acknowledgment number. **Selective reject (SREJ).** If the value of the code subfield is 11, it is an SREJ Sframe. This is a NAK frame used in Selective Repeat ARQ. Note that the HDLC Protocol uses the term selective reject instead of selective repeat. The value of N(R) is the negative acknowledgment number.

Figure 5.23 *Control field format for the different frame types*



Control Field for U-Frames Unnumbered frames are used to exchange session management and control information between connected devices. Unlike S-frames, U-frames contain an information field, but one used for system management information, not user data. As with S-frames, however, much of the information carried by U-frames is contained in codes included in the control field. U-frame codes are divided into two sections: a 2-bit prefix before the P/F bit and a 3-bit suffix after the P/F bit. Together, these two segments (5 bits) can be used to create up to 32 different types of U-frames.