Unit-2

Data Link Layer

Dr. Yogesh Thakare

CSE(AIML)

RCOEM, Nagpur

- The data link layer transforms the physical layer, a raw transmission facility, to a link responsible for node-to-node (hop-to-hop) communication. Specific responsibilities of the data link layer include
- *framing,*
- *addressing,*
- *flow control,*
- *error control*
- *media access control.*

• *Error Detection and Correction*

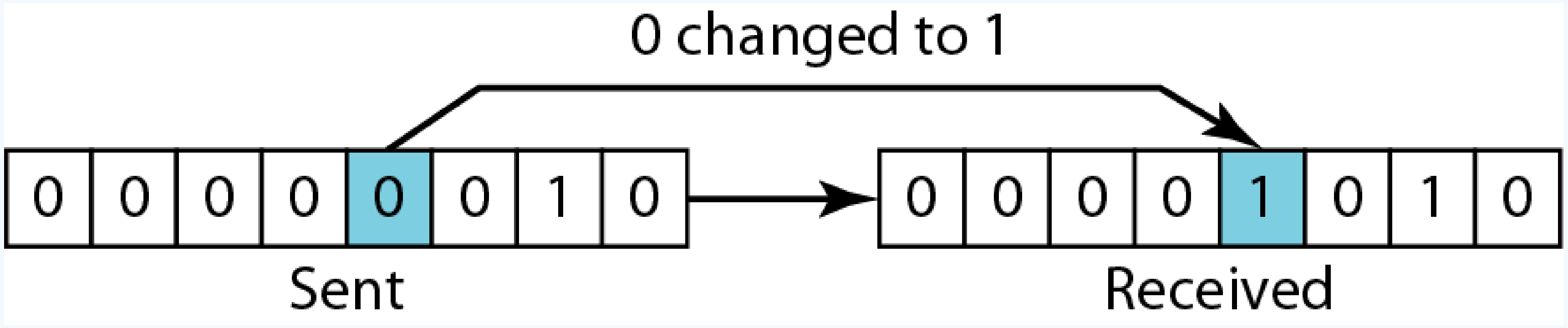
- Networks must be able to transfer data from one device to another with acceptable accuracy.
- For most applications, a system must guarantee that the data received are identical to the data transmitted.
- Any time data are transmitted from one node to the next, they can become corrupted in passage. Many factors can alter one or more bits of a message. Some applications require a mechanism for detecting and correcting errors.

- Types of Errors

1. Single Bit error
2. Burst Error

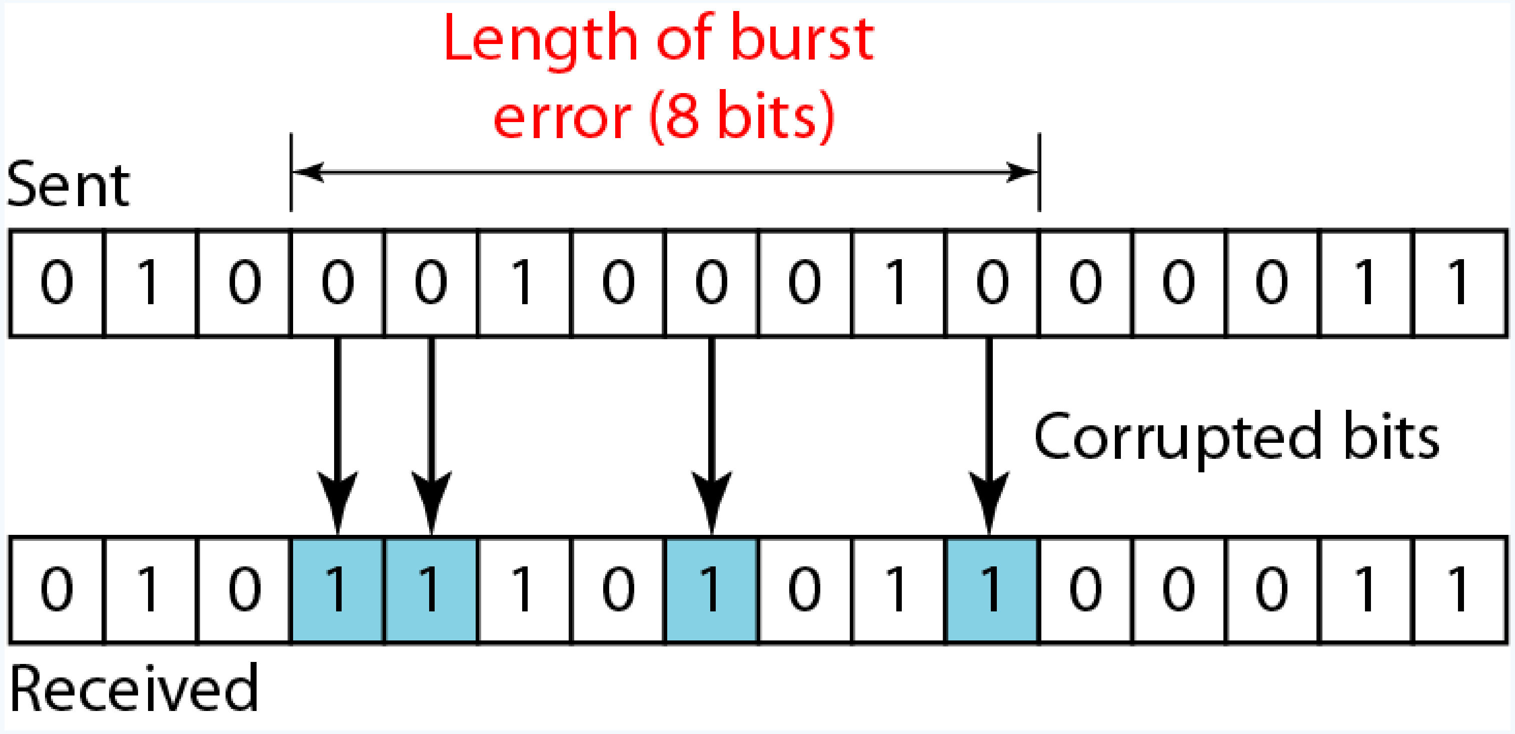
Single Bit error

- 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.



- *Burst Error*

- 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.



- Redundancy

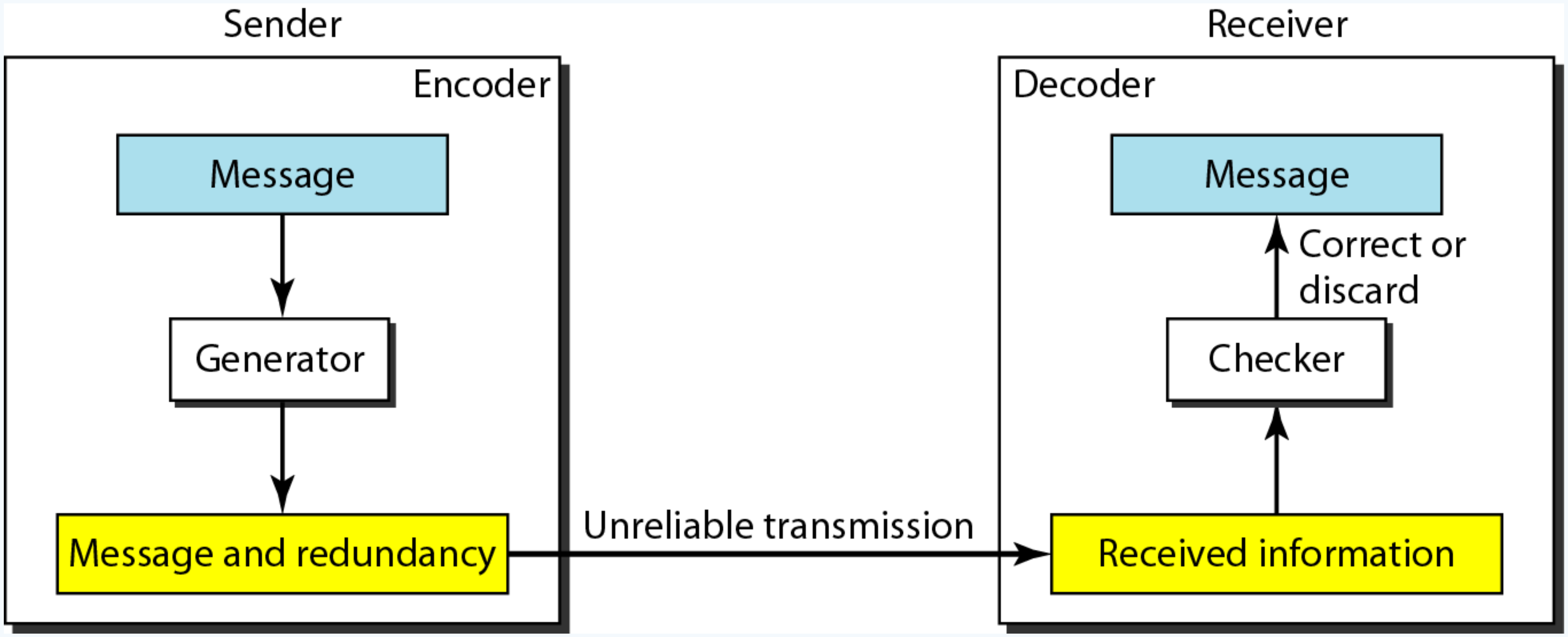
The central concept in detecting or correcting errors is redundancy. To be able to detect or correct errors, we need to send some extra bits with our data. These redundant bits are added by the sender and removed by the receiver. Their presence allows the receiver to detect or correct corrupted bits.

- Detection Versus Correction
- The correction of errors is more difficult than the detection.
- In error 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 one single error in an 8-bit data unit, we need to consider eight possible error locations

- Forward Error Correction Versus Retransmission
- There are two main methods of error correction.
- Forward error correction is the process in which the receiver tries to guess the message by using redundant bits. This is possible, as we see later, if the number of errors is small.
- Correction by retransmission is a technique in which the receiver detects the occurrence of an error and asks the sender to resend the message. Resending is repeated until a message arrives that the receiver believes is error-free.

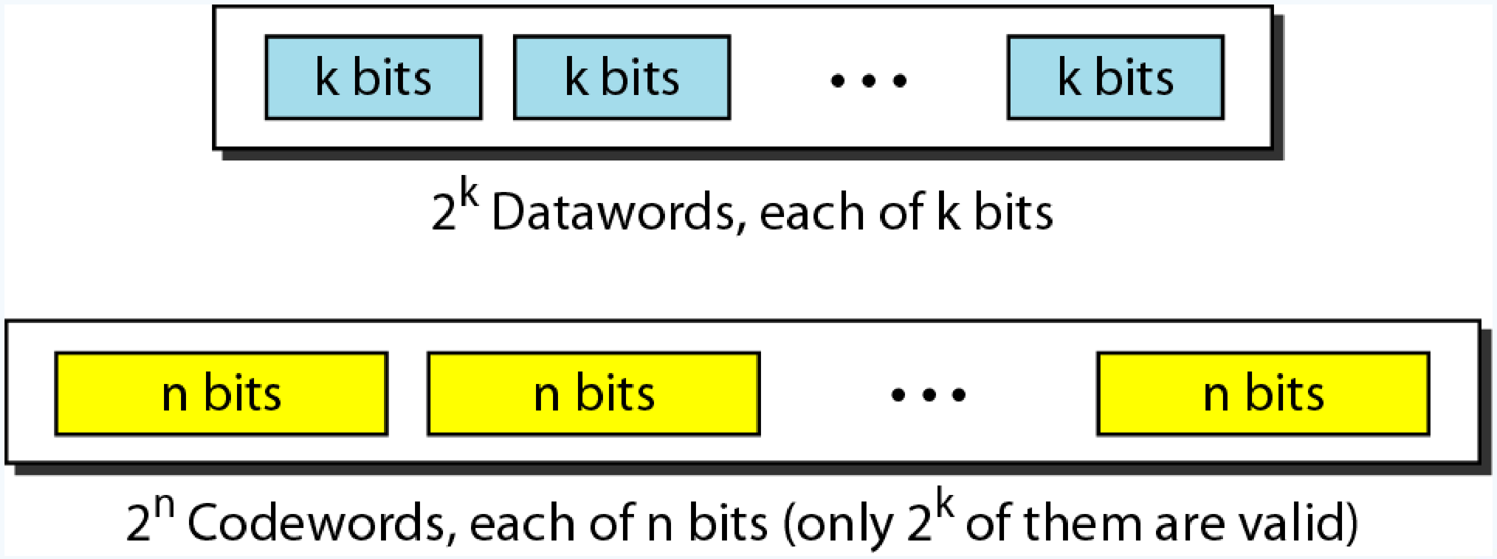
- Coding
- Redundancy is achieved through various coding schemes.
- The sender adds redundant bits through a process that creates a relationship between the redundant bits and the actual data bits.
- The receiver checks the relationships between the two sets of bits to detect or correct the errors. The 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.


k bits
k bits
• • •
k bits
2^k Datawords, each of k bits
n bits
n bits
• • •
n bits
2^n Codewords, each of n bits (only 2^k of them are valid)

- *Example*
- The 4B/5B block coding discussed in Chapter 4 is a good example of this type of coding. In this coding scheme, k = 4 and n = 5. As we saw, we have 2^k = 16 datawords and 2^n = 32 codewords. We saw that 16 out of 32 codewords are used for message transfer and the rest are either used for other purposes or unused.

- **Error Detection**

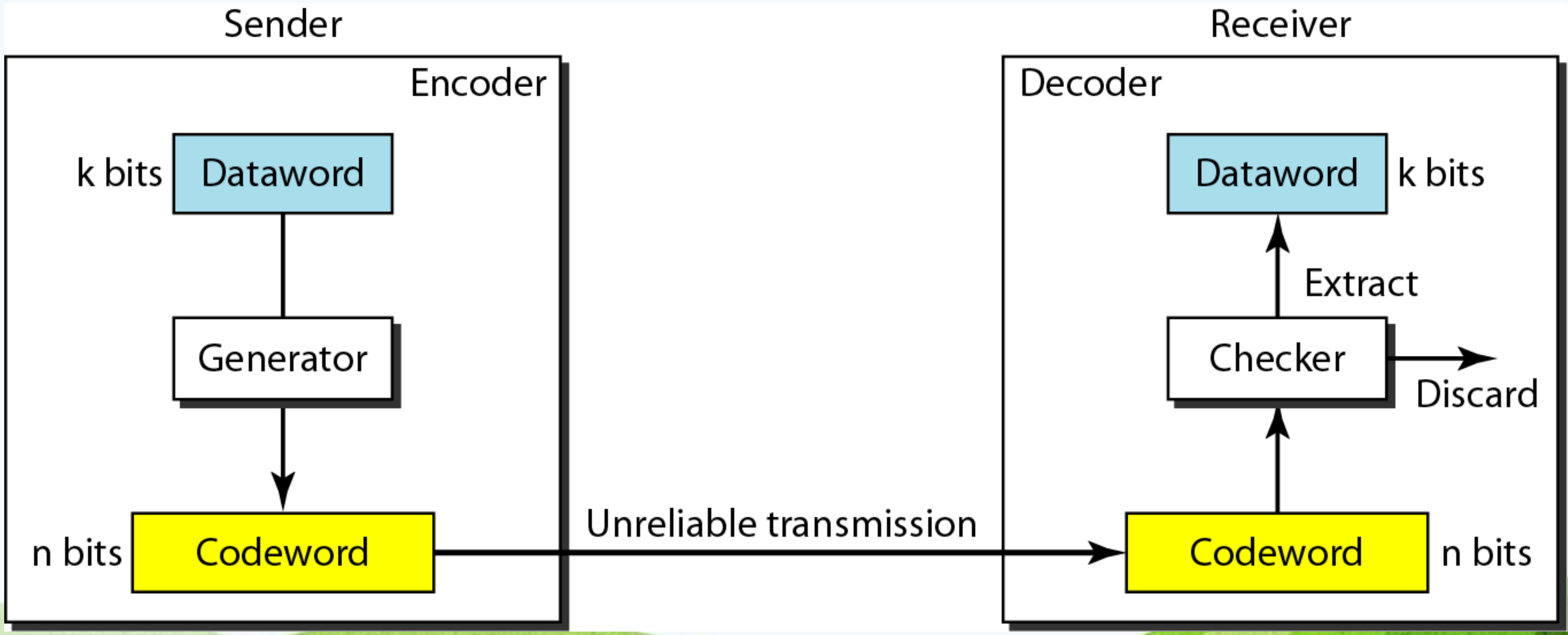
How can errors be detected by using block coding?

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.

- Enough redundancy is added to detect an error.
- The receiver knows an error occurred but does not know which bit(s) is(are) in error.
- Has less overhead than error correction

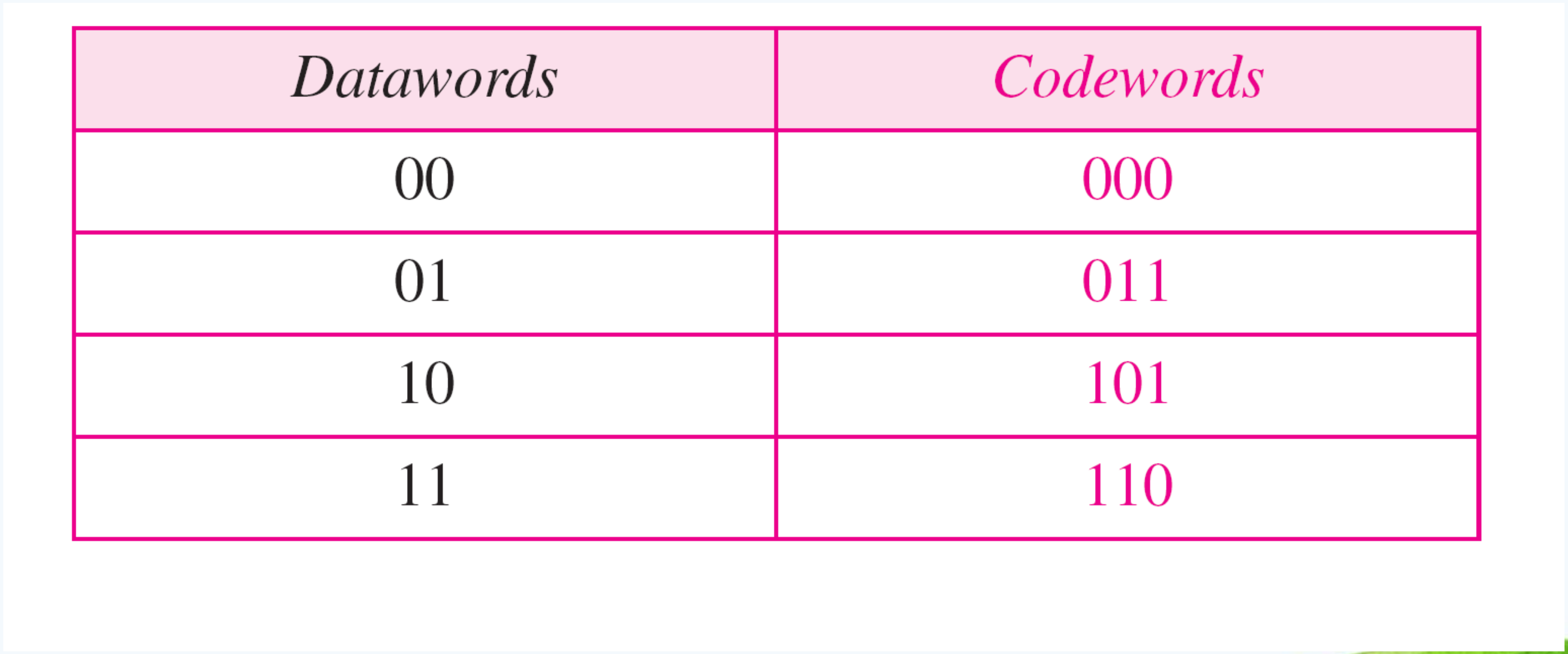


- The sender creates codewords out of datawords by using a generator that applies the rules and procedures of encoding.
- Each codeword sent to the receiver may change during transmission.
- If the received codeword is the same as one of the valid codewords, the word is accepted; the corresponding dataword 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. This type of coding can detect only single errors. Two or more errors may remain undetected.

- *Example*

- Let us assume that *k* =2 and *n* =3. Table 10.1 shows the list of datawords and codewords. Later, we will see how to derive a codeword from a dataword.



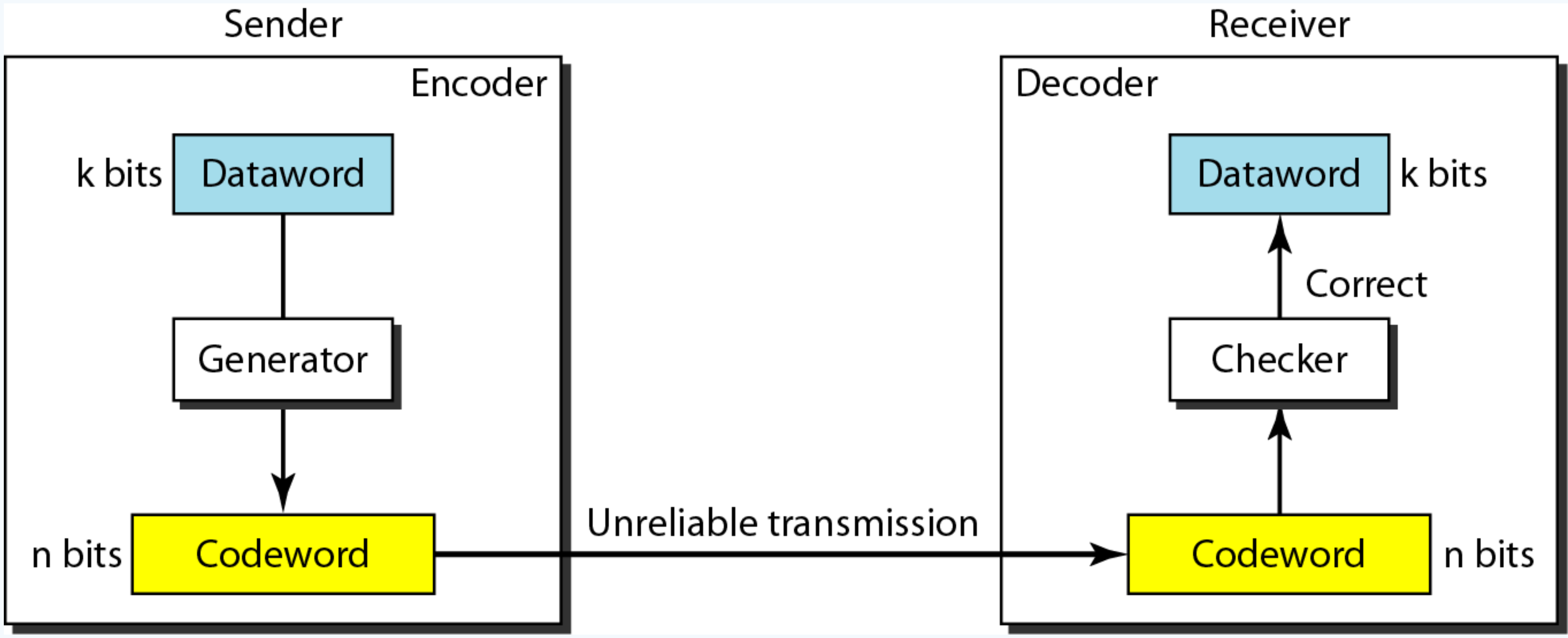
Datawords	*Codewords*
00	000
01	011
10	101
11	110

- 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. This is not a valid codeword and is discarded.
3. The codeword is corrupted during transmission, and 000 is received. This is a valid codeword. The receiver incorrectly extracts the dataword 00. Two corrupted bits have made the error undetectable.

- Error Correction
- As we said before, error correction is much more difficult than error detection.
- In error detection, the receiver needs to know only that the received codeword is invalid; in error correction the receiver needs to find (or guess) the original codeword sent.
- We can say that we need more redundant bits for error correction than for error detection.

- Figure 10.7 shows the role of block coding in error correction. We can see that the idea is the same as error detection but the checker functions are much more complex.



- *We add 3 redundant bits to the 2-bit dataword to make 5-bit codewords. Table 10.2 shows the datawords and codewords. Assume the dataword is 01. The sender creates the codeword 01011. The codeword is corrupted during transmission, and 01001 is received. First, the receiver finds that the received codeword is not in the table. This means an error has occurred. The receiver, assuming that there is only 1 bit corrupted, uses the following strategy to guess the correct dataword.*

Dataword	*Codeword*
00	00000
01	01011
10	10101
11	11110

1. Comparing the received codeword with the first codeword in the table (01001 versus 00000), the receiver decides that the first codeword is not the one that was sent because there are two different bits.

2. By the same reasoning, the original codeword cannot be the third or fourth one in the table.

3. The original codeword must be the second one in the table because this is the only one that differs from the received codeword by 1 bit. The receiver replaces 01001 with 01011 and consults the table to find the dataword 01.

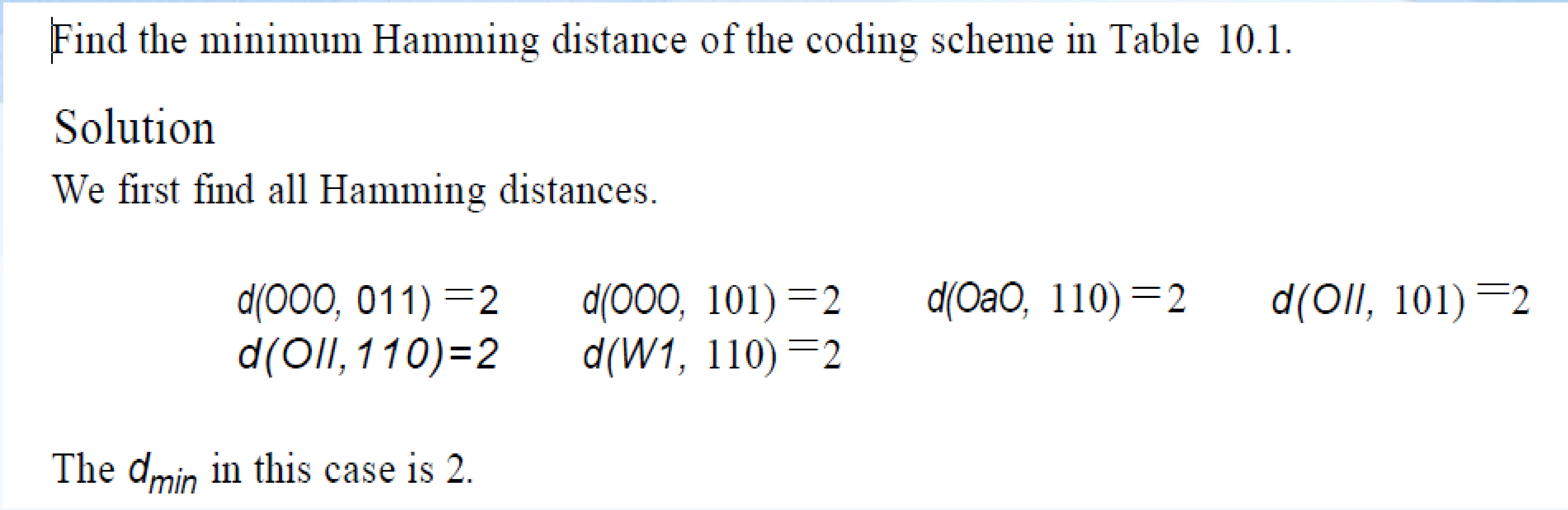
- Hamming Distance
- One of the central concepts in coding for error control is the idea of the Hamming distance.
- The Hamming distance between two words (of the same size) is the number of differences between the corresponding bits. We show the Hamming distance between two words *x* and *y* as *d*(*x*, *y*).
- The Hamming distance can easily be found if we apply the XOR operation $\oplus$ on the two words and count the number of 1's in the result.
- *Note that the Hamming distance is a value greater than zero.*

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

1. The Hamming distance *d(*000, 011) is 2 because 000 XOR 011 is 011 (two 1's).

2. The Hamming distance *d(*10101, 11110) is 3 because 10101 XOR 11110 is 01011 (three 1's).

- Minimum Hamming Distance
- The minimum Hamming distance is the smallest Hamming distance between all possible pairs.
- We use dmin to define the minimum Hamming distance in a coding scheme. To find this value, we find the Hamming distances between all words and select the smallest one.
- Any coding scheme needs to have at least three parameters: the codeword size *n*, the dataword size *k*, and the minimum Hamming distance *dmin*.
- A coding scheme C is written as *C(n, k)* with a separate expression for *dmin*.

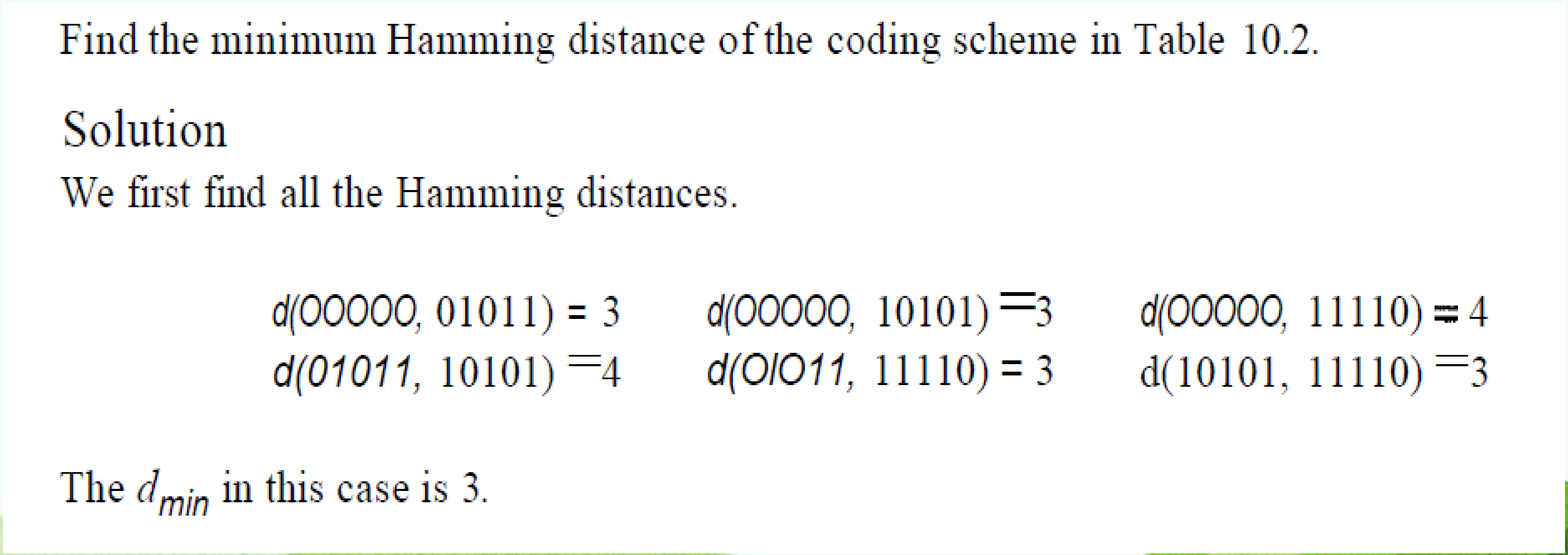

Find the minimum Hamming distance of the coding scheme in Table 10.1.

Solution

We first find all Hamming distances.

d(OOO, 011) =2 d(OOO, 101) =2 d(OaO, 110) =2 d(OII, 101) =2
d(OII, 110)=2 d(W1, 110) =2

The d_{min} in this case is 2.


Find the minimum Hamming distance of the coding scheme in Table 10.2.

Solution

We first find all the Hamming distances.

d(OOOOO, 01011) = 3 d(OOOOO, 10101) =3 d(OOOOO, 11110) = 4
d(01011, 10101) =4 d(OlO11, 11110) = 3 d(10101, 11110) =3

The d_{min} in this case is 3.

What is the Hamming distance for each of the following codewords:

a. d (10000, 00000)

b. d (10101, 10000)

c. d (11111,11111)

d. d (000, 000)

Answer:-

a. d (10000, 00000) = 1

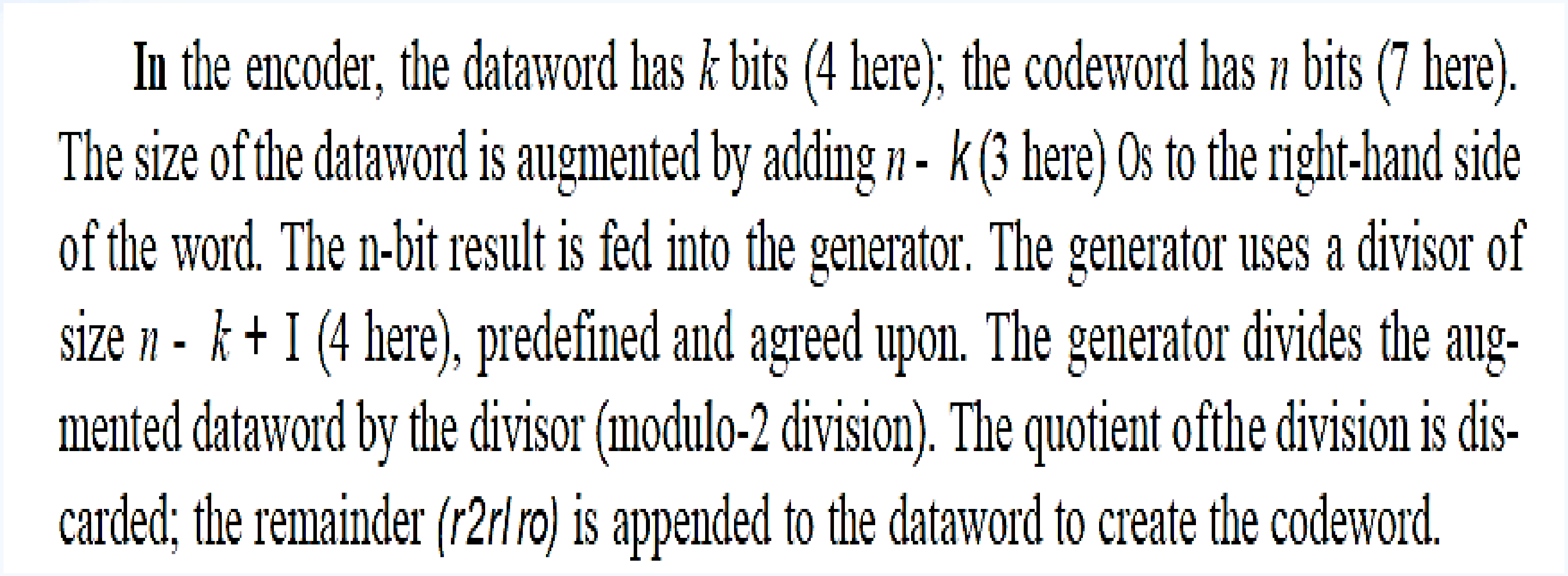
b. d (10101, 10000) = 2

c. d (1111, 1111) = 0

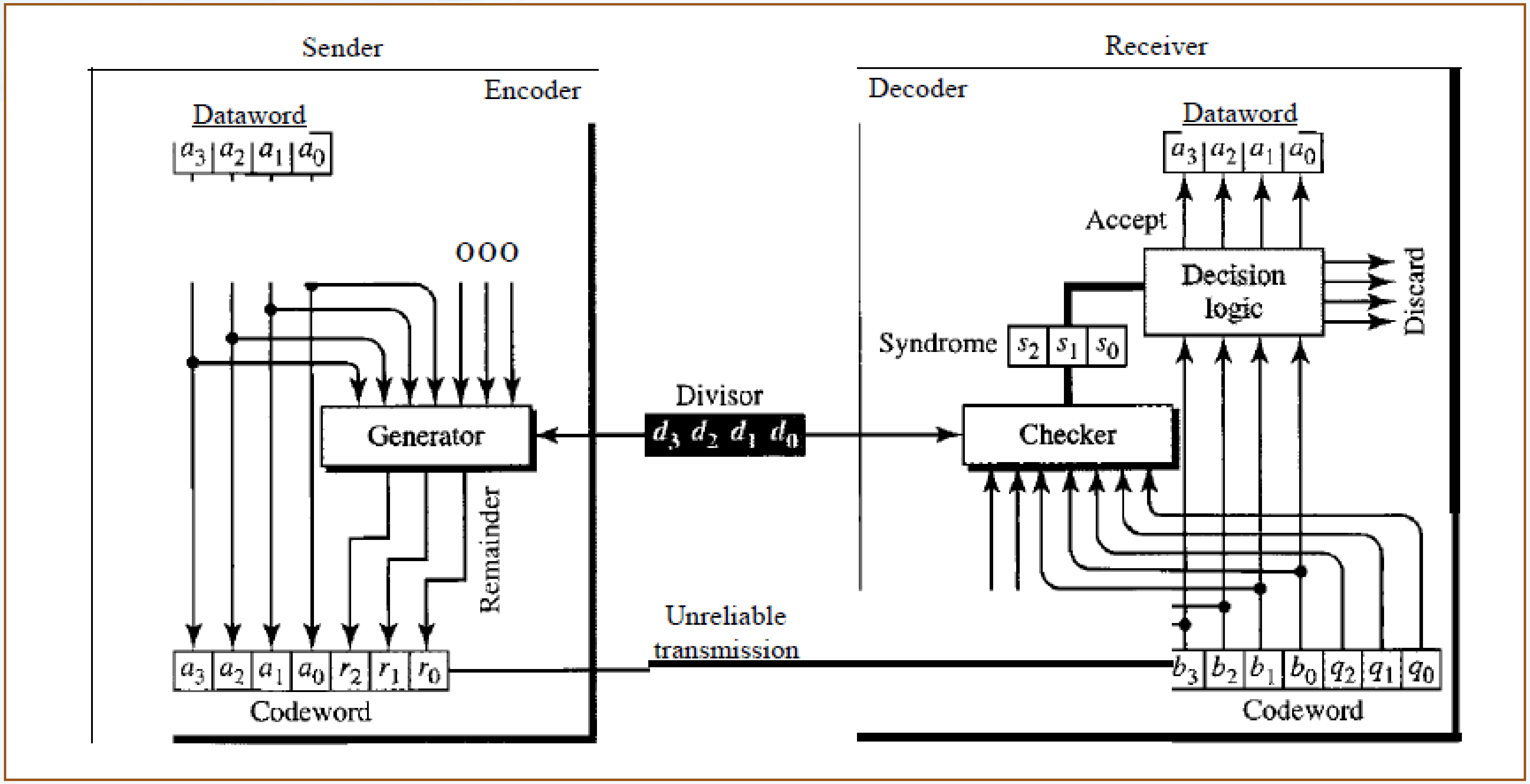
d. d (000, 000) = 0

- 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.

- Cyclic Redundancy Check
- Encoder:-



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* + I (4 here), predefined and agreed upon. The generator divides the augmented dataword by the divisor (modulo-2 division). The quotient ofthe division is discarded; the remainder *(r2rl ro)* is appended to the dataword to create the codeword.


Sender
Encoder
Dataword
a_3 a_2 a_1 a_0
000
Generator
Remainder
a_3 a_2 a_1 a_0 r_2 r_1 r_0
Codeword
Divisor
d_3 d_2 d_1 d_0
Unreliable transmission
Receiver
Decoder
Dataword
a_3 a_2 a_1 a_0
Accept
Decision logic
Discard
Syndrome
s_2 s_1 s_0
Checker
b_3 b_2 b_1 b_0 q_2 q_1 q_0
Codeword

- Decoder:-
- The decoder receives the possibly corrupted codeword. 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 as, the 4 leftmost bits of the codeword are accepted as the dataword (interpreted as no error); otherwise, the 4 bits are discarded (error).

- Flow Control & Error Control Protocols
- Data communication requires at least two devices working together, one to send and the other to receive.
- Even such a basic arrangement requires a great deal of coordination for an intelligible exchange to occur.
- The most important responsibilities of the data link layer are flow control and error control. Collectively, these functions are known as data link control.

- Flow Control

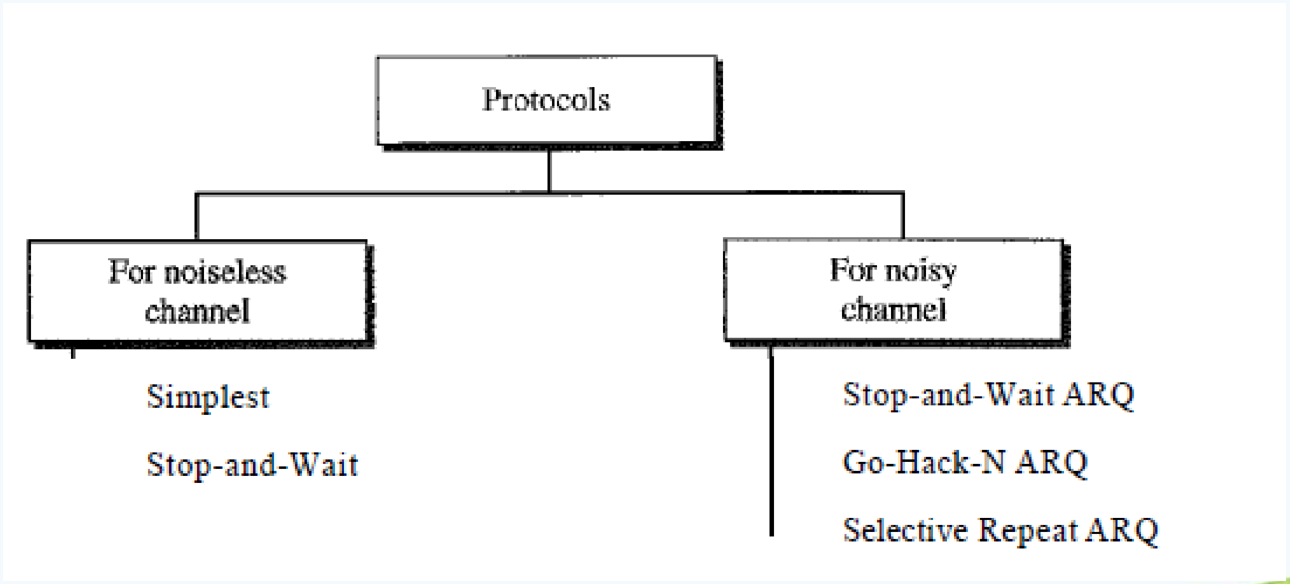
- Flow control coordinates the amount of data that can be sent before receiving an acknowledgment.
- In most protocols,flow control is a set of procedures that tells the sender how much data it can transmit before it must wait for an acknowledgment from the receiver.
- The flow of data must not be allowed to overwhelm the receiver.

- Any receiving device has a limited speed at which it can process incoming data and a limited amount of memory in which to store incoming data.
- The receiving device must be able to inform the sending device before those limits are reached and to request that the transmitting device send fewer frames or stoptemporarily.
- The rate of such processing is often slower than the rate of transmission. For this reason, each receiving device has a block of memory, called a buffer, reserved for storing incoming data until they are processed.

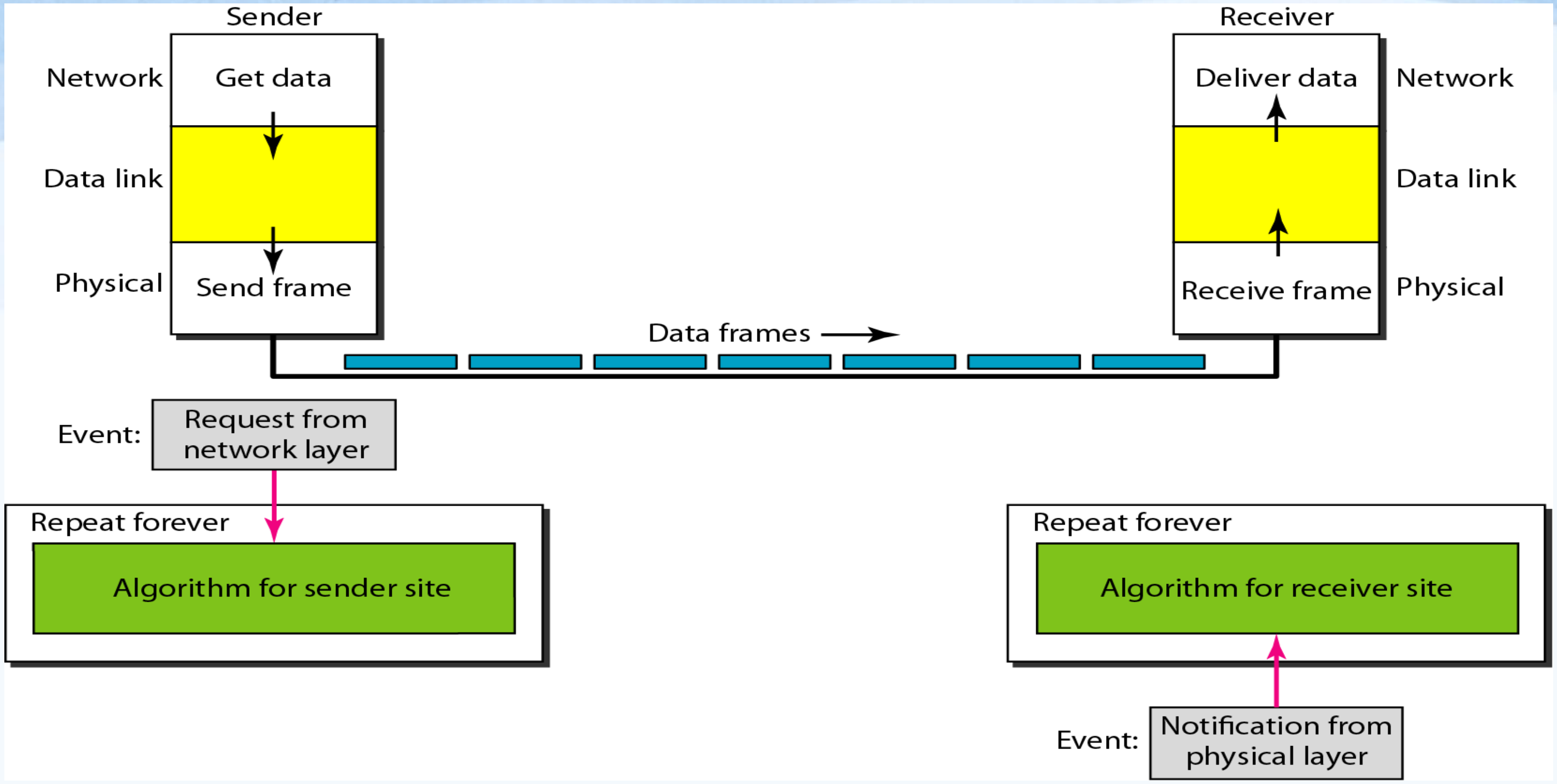
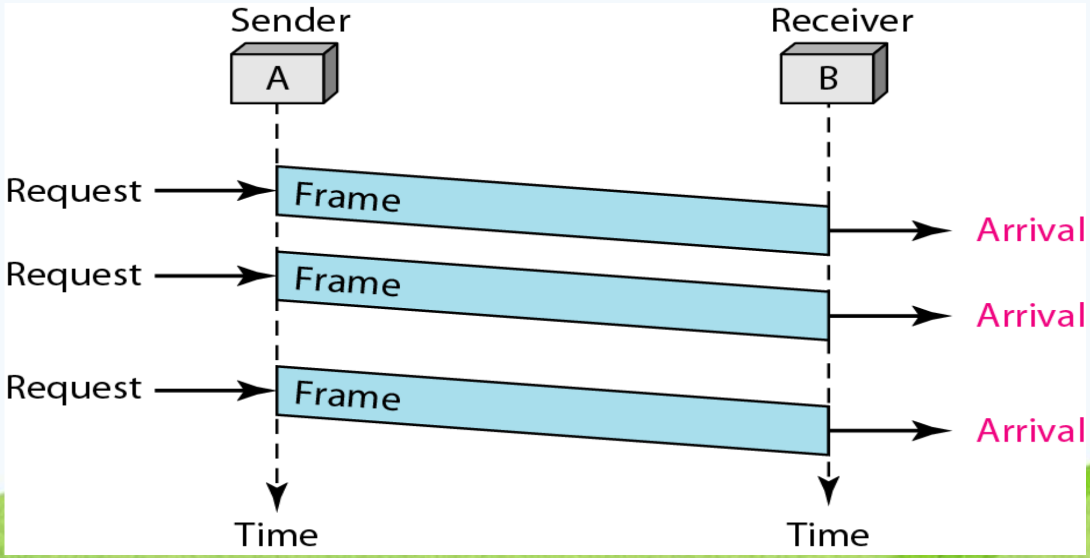
• Error Control

- 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, specified frames are retransmitted. This process is called automatic repeat request (ARQ).
- PROTOCOLS

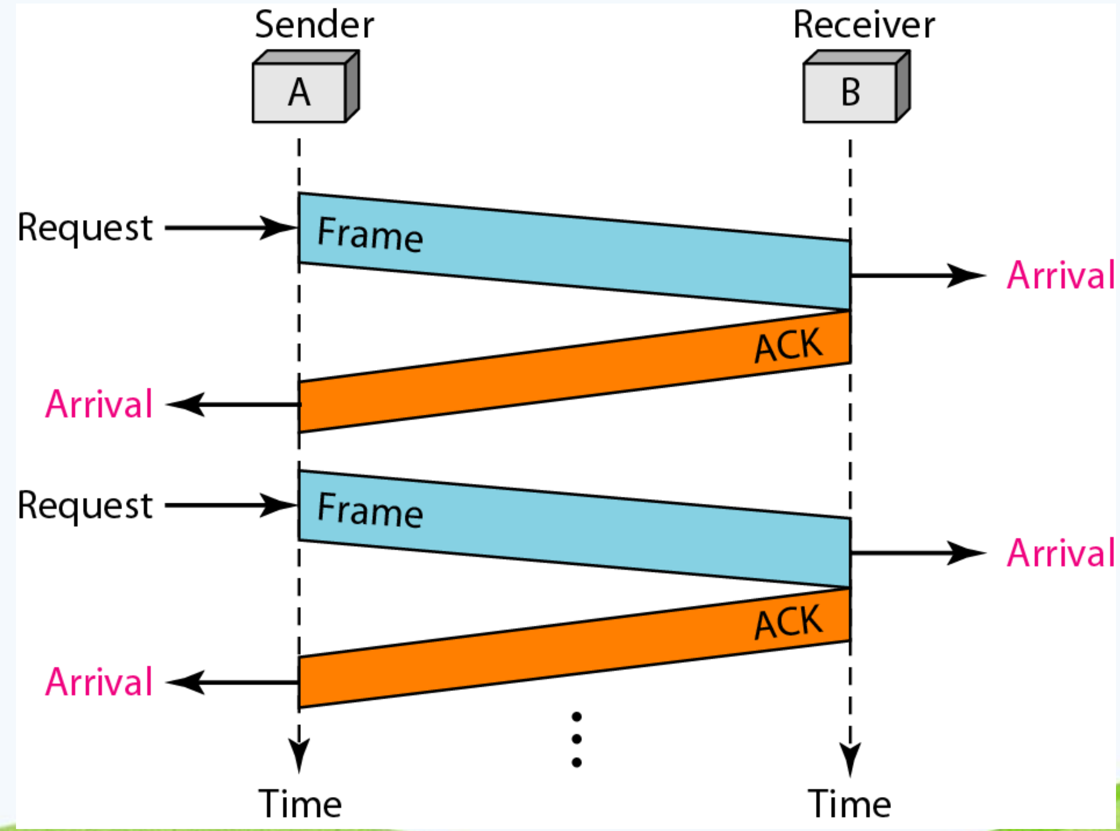


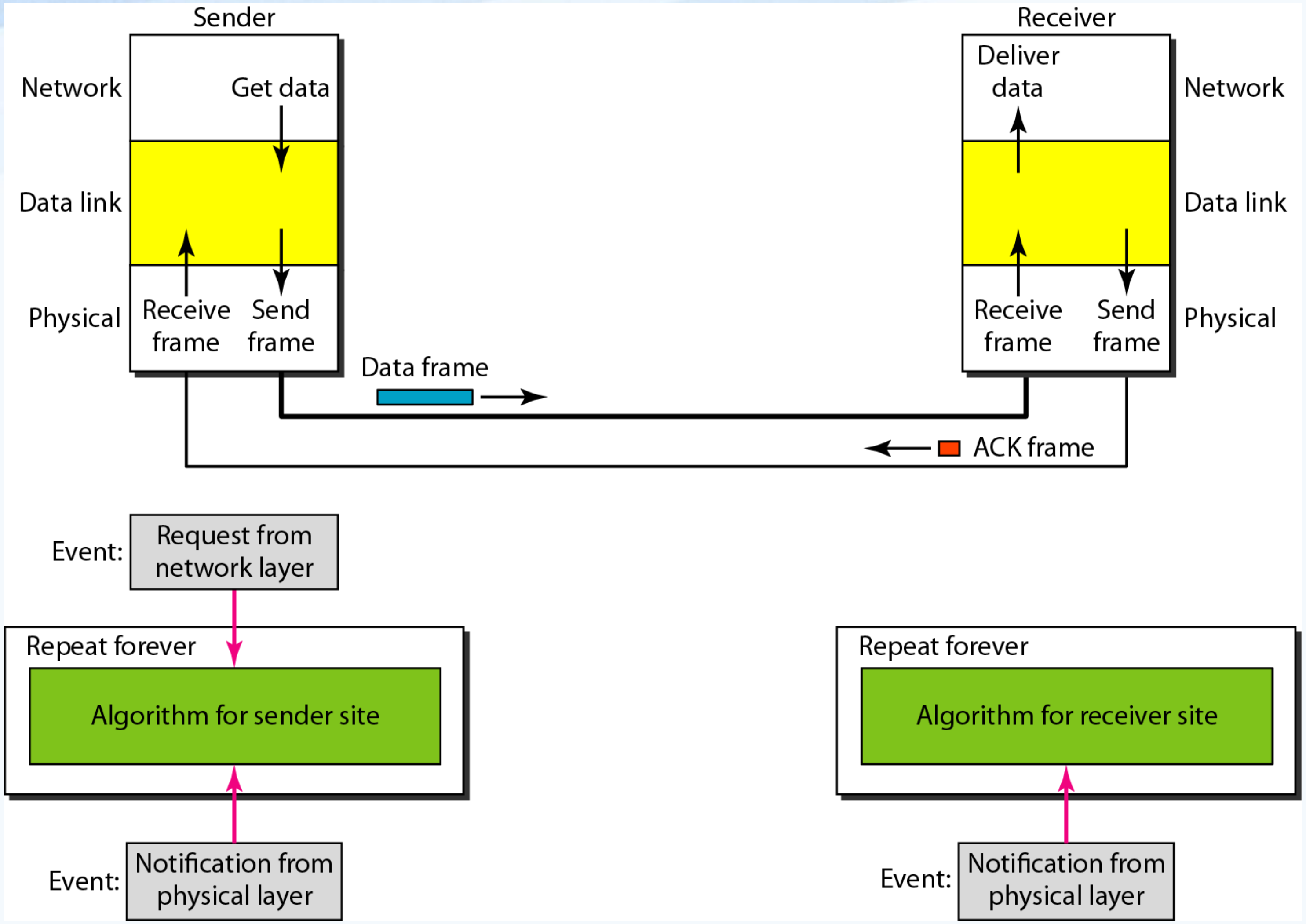
- Simplest
- In simplest protocol no flow and error control mechanism is available.
- It is a unidirectional protocol in which data frames are traveling in only one direction-from the sender to receiver.
- The data link layer at the sender site gets data from its network layer, makes a frame out of the data, and sends it.
- The data link layer at the receiver site receives a frame from its physical layer, extracts data from the frame, and delivers the data to its network layer.


Sender
Receiver
Network
Get data
Deliver data
Network
Data link
Data link
Physical
Send frame
Receive frame
Physical
Data frames
Event:
Request from network layer
Repeat forever
Algorithm for sender site
Repeat forever
Algorithm for receiver site
Event:
Notification from physical layer

Sender
Receiver
A
B
Request
Frame
Arrival
Request
Frame
Arrival
Request
Frame
Arrival
Time
Time

- Stop-and-Wait Protocol
- If data frames arrive at the receiver site faster than they can be processed, the frames must be stored until their use.
- Normally, the receiver does not have enough storage space, especially if it is receiving data from many sources.
- This may result in either the discarding of frames or denial of service. To prevent the receiver from becoming overwhelmed with frames, we somehow need to tell the sender to slow down. There must be feedback from the receiver to the sender.

- The protocol we discuss now is called the Stop-and-Wait Protocol because the sender sends one frame, stops until it receives confirmation from the receiver (okay to go ahead), and then sends the next frame.




Sender
Network
Get data
Data link
Physical
Receive frame
Send frame
Data frame
Receiver
Deliver data
Network
Data link
Physical
Receive frame
Send frame
ACK frame
Event:
Request from network layer
Repeat forever
Algorithm for sender site
Event:
Notification from physical layer
Repeat forever
Algorithm for receiver site
Event:
Notification from physical layer

• Advantage

- This protocol provides flow control.
- One of the main advantages of the stop and wait protocol is its simplicity.
- If we increase the data packet size, the efficiency is going to increase. Hence, it is suitable for transmitting big data packets.
- **Stop & wait** protocol is accurate as the sender sends the next frame to the receiver only when the acknowledgment of the previous packet is received. So there is less chance of the frame being lost.

- *Disadvantage*
- **Problems Occur Due to Lost Data**
- Suppose the sender sends the data packet, but the data packet is lost due to some reason. Since the receiver has not received the packet for a long time, so the sender does not receive any acknowledgment from the receiver, so it will not send the next packet.
- **In this case, two problems occur in data transmission:**
- For an acknowledgment, the sender has to wait for an infinite amount of time.
- The receiver will also have to wait for an infinite amount of time to receive the data.

- ## Stop & Wait ARQ

- Stop-and-Wait Automatic Repeat Request (Stop-and Wait ARQ), adds a simple error control mechanism to the Stop-and-Wait Protocol. Let us see how this protocol detects and corrects errors.

- To detect and correct corrupted frames, we need to add redundancy bits to our data frame.

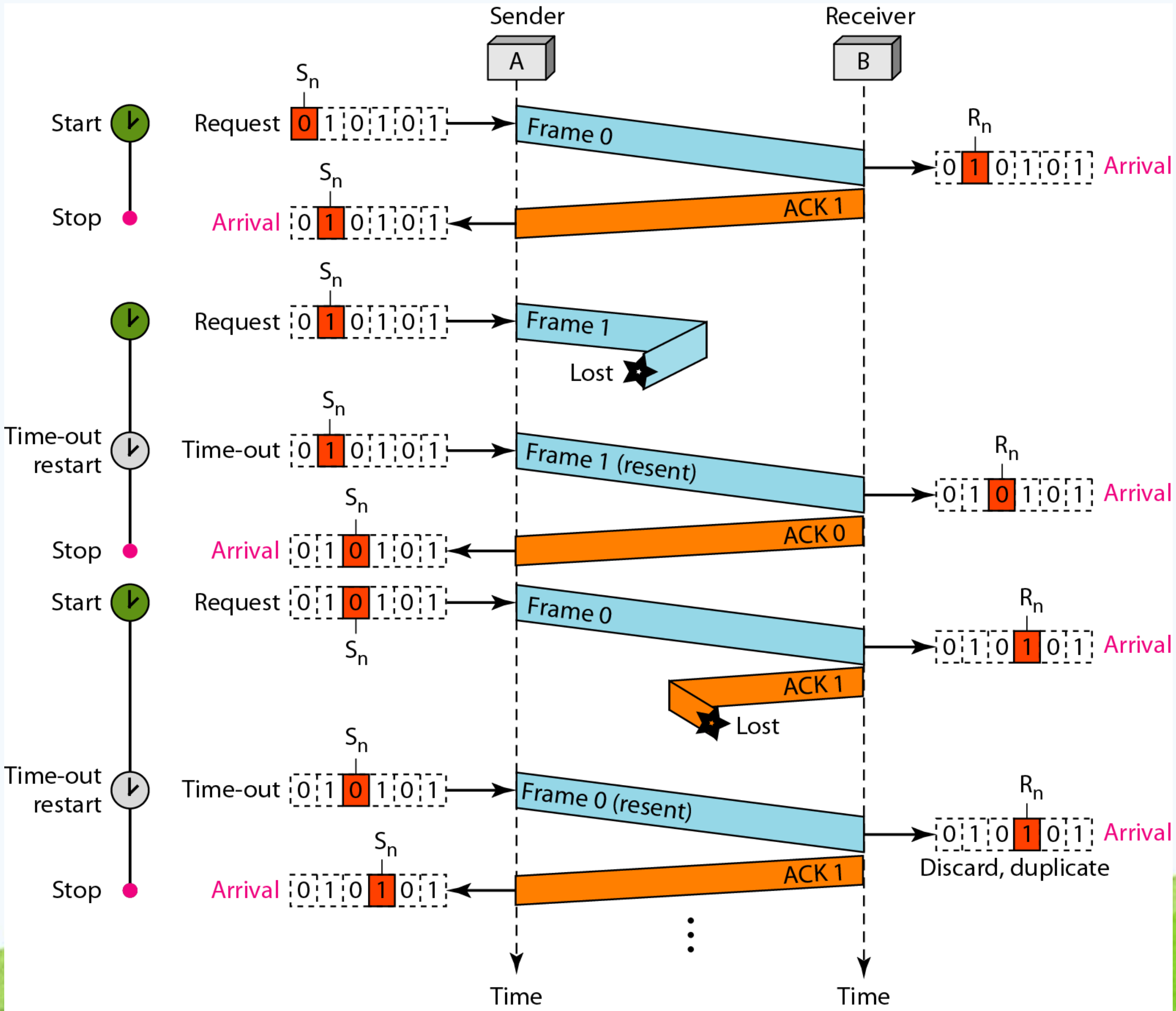
- When the frame arrives at the receiver site, it is checked and if it is corrupted, it is silently discarded.

- Lost frames are more difficult to handle than corrupted ones.

- In our previous protocols, there was no way to identify a frame. The received frame could be the correct one, or a duplicate, or a frame out of order. The solution is to number the frames.
- When the receiver receives a data frame that is out of order, this means that frames were either lost or duplicated.
- If the receiver does not respond when there is an error, how can the sender know which frame to resend?

- To remedy this problem, the sender keeps a copy of the sent frame.
- At the same time, it starts a timer.
- If the timer expires and there is no ACK for the sent frame, the frame is resent, the copy is held, and the timer is restarted.
- Since the protocol uses the stop-and-wait mechanism, there is only one specific frame that needs an ACK even though several copies of the same frame can be in the network.

- In Stop-and-Wait ARQ, we use sequence numbers to number the frames. The sequence numbers are based on modulo-2 arithmetic.



- Since an ACK frame can also be corrupted and lost, it too needs redundancy bits and a sequence number.
- The ACK frame for this protocol has a sequence number field. In this protocol, the sender simply discards a corrupted ACK frame or ignores an out-of-order one.
- Since the sequence numbers must be suitable for both data frames and ACK frames, we use this convention: The acknowledgment numbers always announce the sequence number of the next frame expected by the receiver. For example, if frame 0 has arrived safe and sound, the receiver sends an ACK frame with acknowledgment 1 (meaning frame 1 is expected next). If frame 1 has arrived safe and sound, the receiver sends an ACK frame with acknowledgment 0 (meaning frame 0 is expected).

- **Advantages of Stop and Wait ARQ**

- The stop and wait ARQ can be used in both the data link layer and the transport layer.
- The stop and wait ARQ provides both error management and flow control management.
- The stop and wait ARQ works in half-duplex mode as at a time only the sender can send the message or the receiver can send the ACK.
- The stop and wait ARQ ensures that the information is received by the receiver in the order it was sent.

- *Disadvantage*

- The data can be lost in between the transmission. So, in such a case, the sender waits for ACK and the receiver waits for the data frame for an infinite amount of time.
- The ACK from the receiver may get lost in the channel. So, the sender waits for ACK for an infinite amount of time.
- The window size of the sender and the receiver is only 1. So, only one frame can be sent at a time.
- As there is a timer concept, so the sender must wait for a long duration before retransmission. Hence, the stop and wait ARQ is a slow protocol.

Example 11.4

Assume that, in a Stop-and-Wait ARQ system, the bandwidth of the line is 1 Mbps, and 1 bit takes 20 ms to make a round trip. What is the bandwidth-delay product? If the system data frames are 1000 bits in length, what is the utilization percentage of the link?

Solution

The bandwidth-delay product is

$$\boxed{(1 \times 10^6) \times (20 \times 10^{-3}) = 20{,}000 \text{ bits}}$$

Example 11.4 (continued)

The system can send 20,000 bits during the time it takes for the data to go from the sender to the receiver and then back again. However, the system sends only 1000 bits. We can say that the link utilization is only 1000/20,000, or 5 percent. For this reason, for a link with a high bandwidth or long delay, the use of Stop-and-Wait ARQ wastes the capacity of the link.

A system uses the Stop-and-Wait Protocol. If each packet carries 1000 bits of data, how long does it take to send 1 million bits of data if the distance between the sender and receiver is 5000 Km and the propagation speed is 2 x 10^8 m/s? Ignore transmission, waiting, and processing delays. We assume no data or control frame is lost or damaged.

Answer:-

Packet size = 10^3 bits
Total data to be transmitted = 10^6 bits
Therefore,
number of packets to be transmitted = $10^6 / 10^3$= 1000
Propagation delay for one packet
= Distance / Propagation speed
= $(5 * 10^6) / (2*10^8)$ = 0.025 s
Propagation delay for one ACK = Distance / Propagation speed = $(5 * 10^6) / (2*10^8)$ = 0.025 s
Here, transmission delay is 0.
So, Total time to transmit one packet and receive its ACK
= 2 * 0.025 = 0.05 s
Therefore, total time to transmit 1000 packets = 1000 * 0.05 = **50 s**

11-1 FRAMING

The data link layer 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.

Topics discussed in this section:

Fixed-Size Framing
Variable-Size Framing

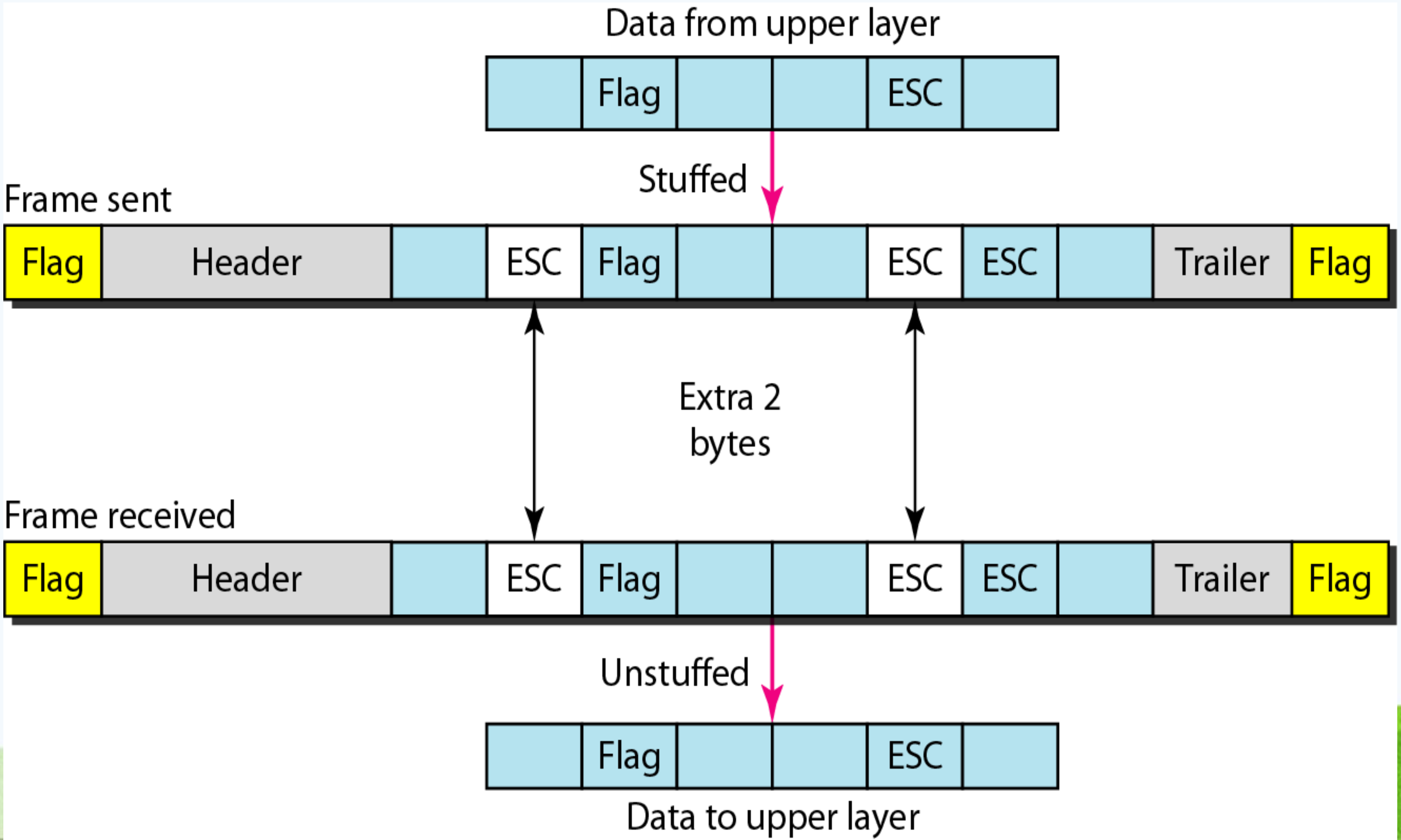
Byte stuffing -destuffing

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), which 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 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 flag? The receiver removes the escape character, but keeps the flag, which is incorrectly interpreted as the end of the frame.



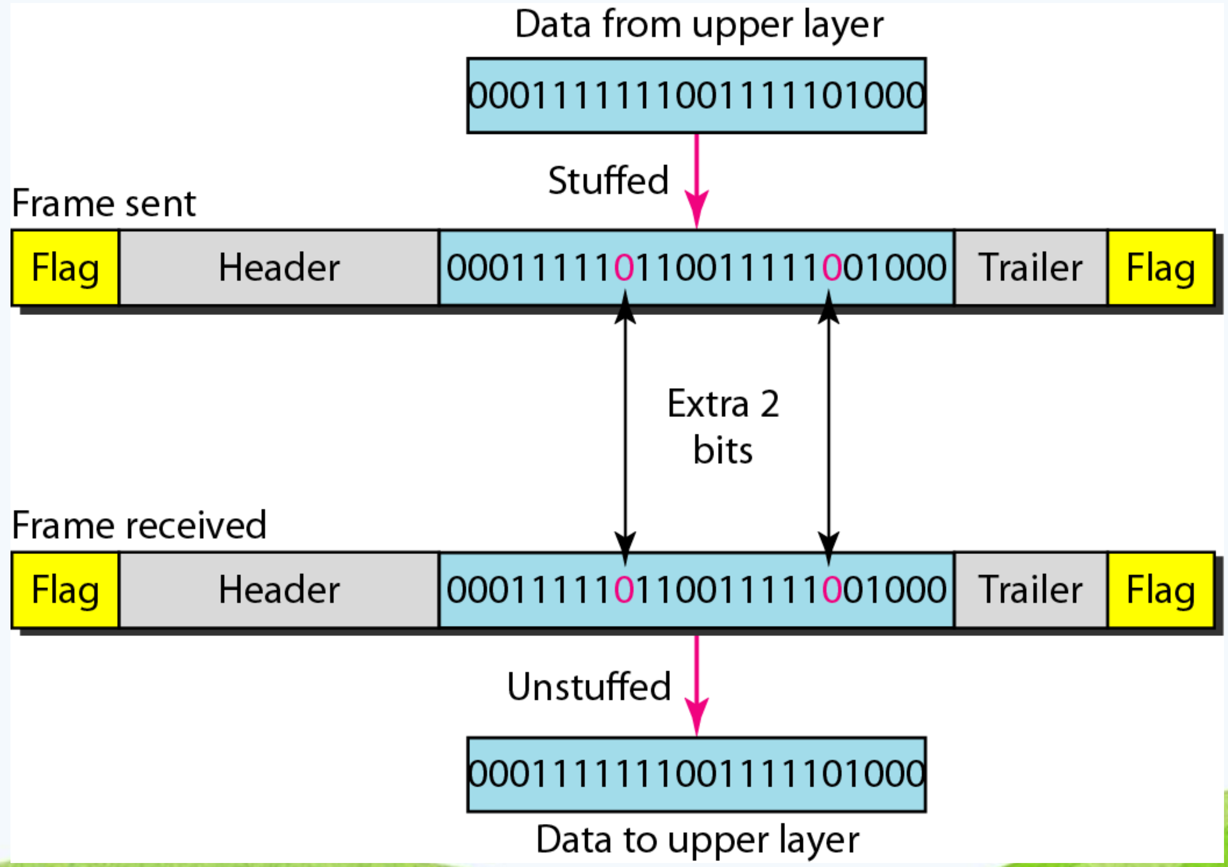
Bit Stuffing

In bit 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.

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.

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



Go-Back-N Automatic Repeat Request

To improve the efficiency of transmission (filling the pipe), multiple frames must be in transition while waiting for acknowledgment.
In other words, we need to let more than one frame be outstanding to keep the channel busy while the sender is waiting for acknowledgment.

The first is called Go-Back-N Automatic Repeat Request.

In this protocol we can send several frames before receiving acknowledgments; we keep a copy of these frames until the acknowledgments arrive.

In the Go-Back-N Protocol, the sequence numbers are modulo 2^m, where *m* is the size of the sequence number field in bits.

If the header of the frame allows *m* bits for the sequence number, the sequence numbers range from 0 to *2^m*- 1. For example, if *m* is 4, the only sequence numbers are 0 through 15 inclusive. However, we can repeat the sequence. So the sequence numbers are

0, 1,2,3,4,5,6, 7,8,9, 10, 11, 12, 13, 14, 15,0, 1,2,3,4,5,6,7,8,9,10, 11, ...

In other words, the sequence numbers are modulo-2^m.

.

Sliding Window

The sliding window is an abstract concept that defines the range of sequence numbers that is the concern of the sender and receiver.
In other words, the sender and receiver need to deal with only part of the possible sequence numbers.
The range which is the concern of the sender is called the send sliding window; the range that is the concern of the receiver is called the receive sliding window.

The send window is an imaginary box covering the sequence numbers of the data frames which can be in transit.
In each window position, some of these sequence numbers define the frames that have been sent; others define those that can be sent. The maximum size of the window is *2^m* – 1.

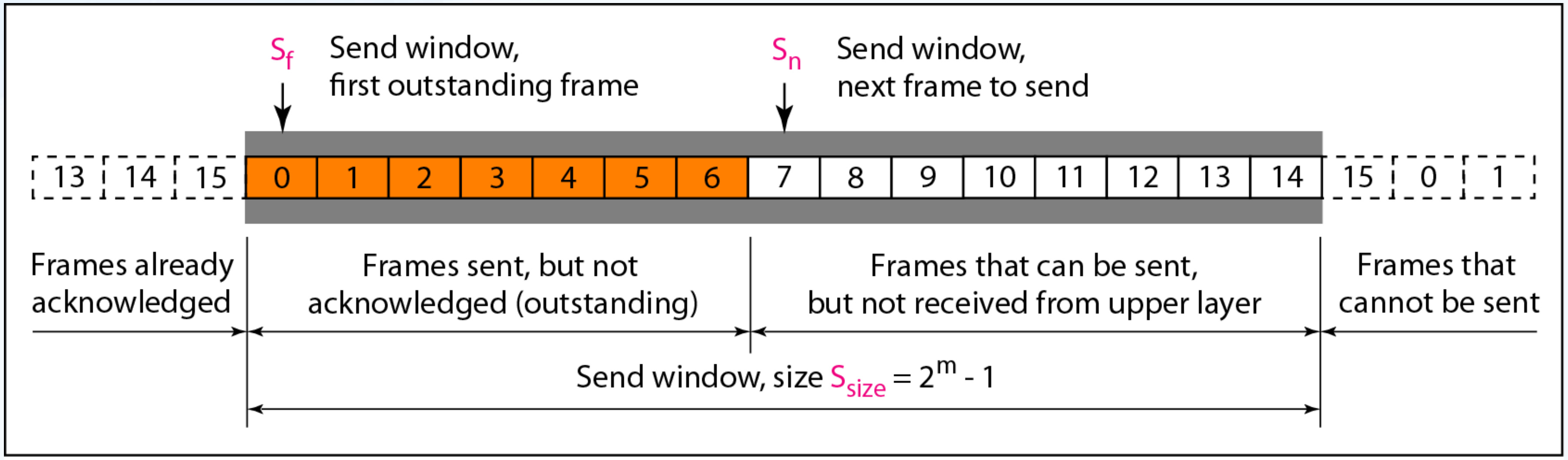
The window at any time divides the possible sequence numbers into four regions.

The first region, from the far left to the left wall of the window, defines the sequence numbers belonging to frames that are already acknowledged. The sender does not worry about these frames and keeps no copies of them.

The second region defines the range of sequence numbers belonging to the frames that are sent and have an unknown status. The sender needs to wait to find out if these frames have been received or were lost. We call these outstanding frames.

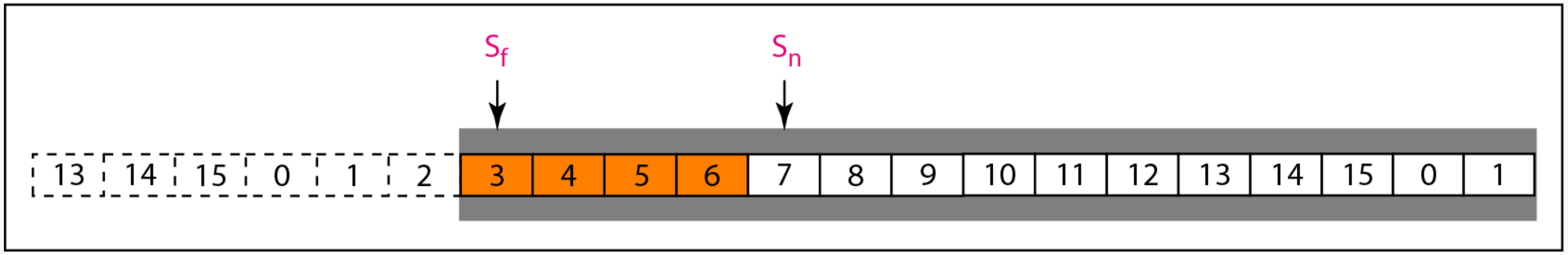
The third range defines the range of sequence numbers for frames that can be sent; however, the corresponding data packets have not yet been received from the network layer.

Finally, the fourth region defines sequence numbers that cannot be used until the window slides.



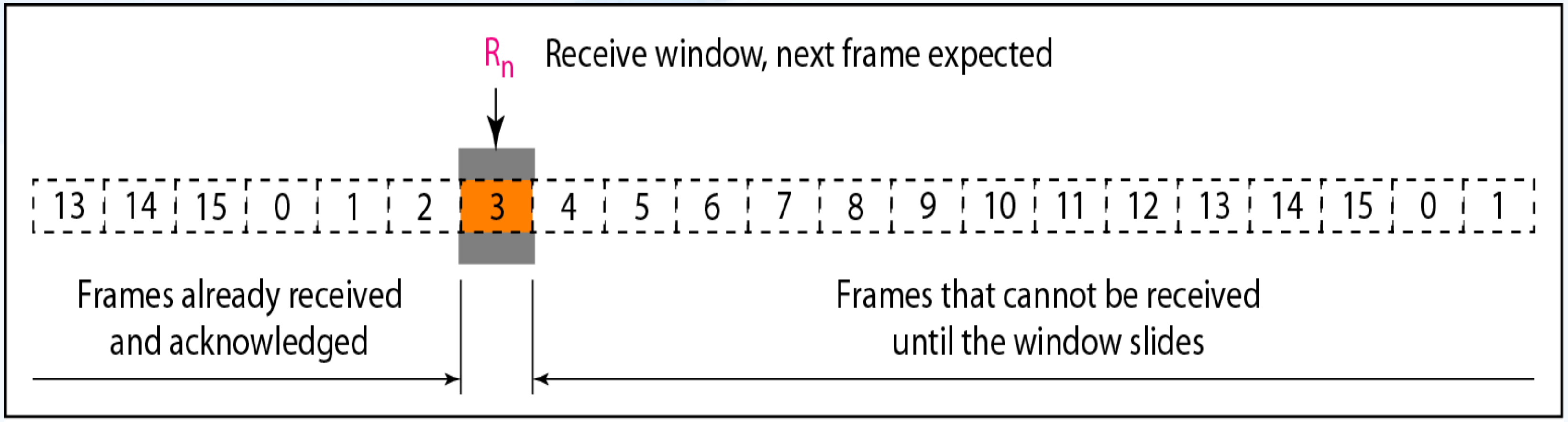
S_f
Send window, first outstanding frame
S_n
Send window, next frame to send
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
Frames already acknowledged
Frames sent, but not acknowledged (outstanding)
Frames that can be sent, but not received from upper layer
Frames that cannot be sent
Send window, size $S_{size} = 2^m - 1$

a. Send window before sliding

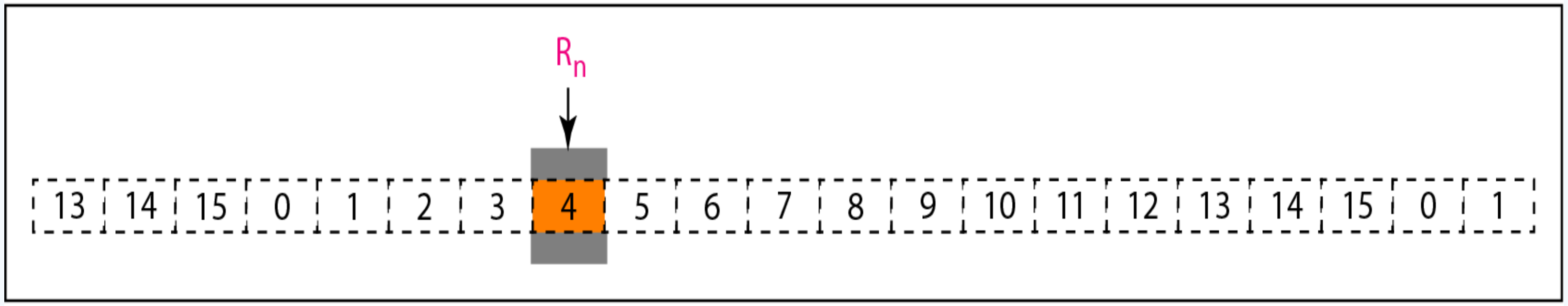


b. Send window after sliding

The window itself is an abstraction; three variables define its size and location at any time. We call these variables Sf (send window, the first outstanding frame), Sn (send window, the next frame to be sent), and Ssize (send window, size).
The variable Sf defines the sequence number of the first (oldest) outstanding frame.
The variable Sn holds the sequence number that will be assigned to the next frame to be sent. Finally, the variable Ssize defines the size of the window, which is fixed in our protocol.


R_n
Receive window, next frame expected
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
Frames already received and acknowledged
Frames that cannot be received until the window slides

a. Receive window

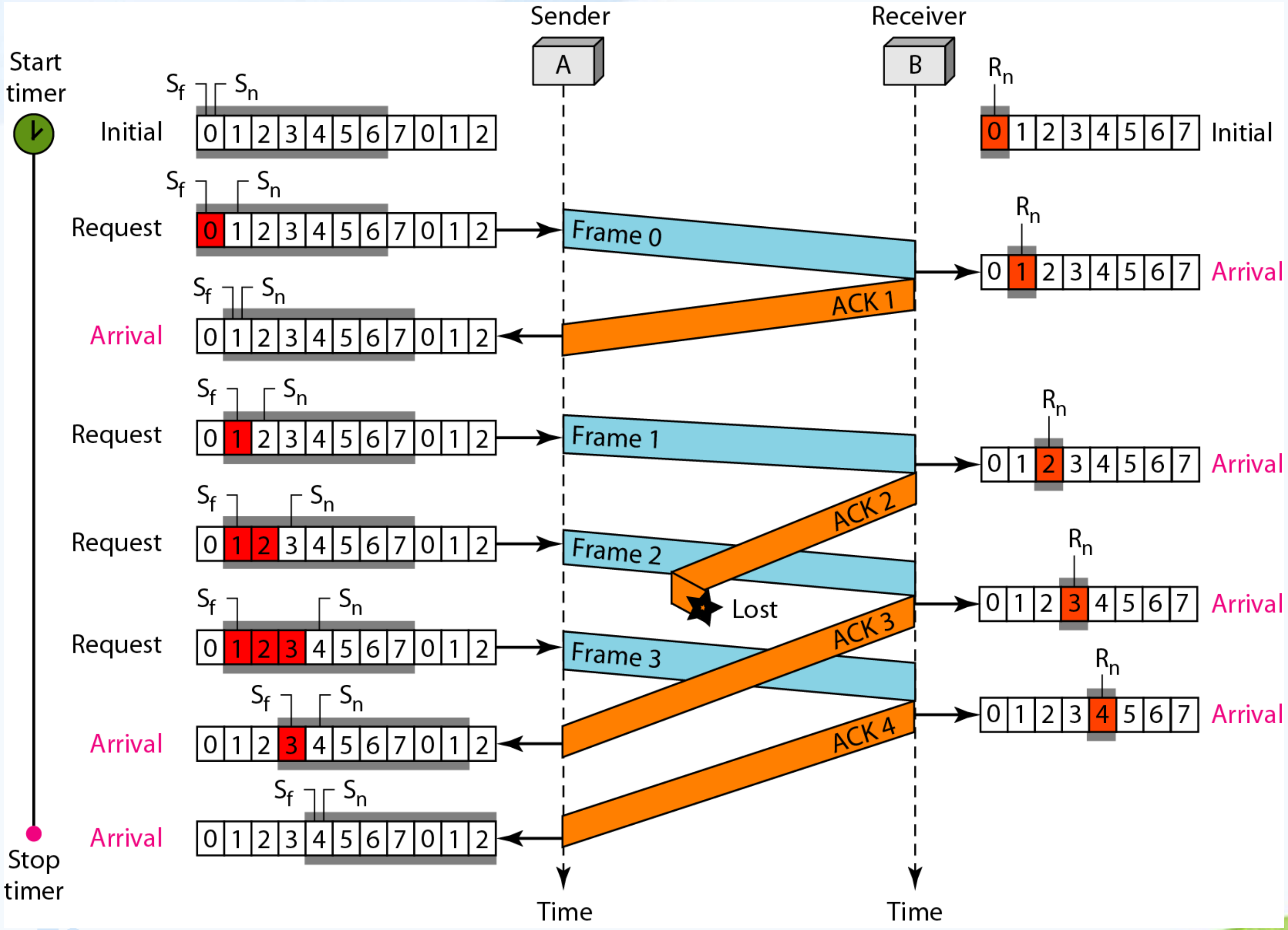


b. Window after sliding

Example 11.6

Figure 11.16 shows an example of Go-Back-N. This is an example of a case where the forward channel is reliable, but the reverse is not. No data frames are lost, but some ACKs are delayed and one is lost. The example also shows how cumulative acknowledgments can help if acknowledgments are delayed or lost. After initialization, there are seven sender events. Request events are triggered by data from the network layer; arrival events are triggered by acknowledgments from the physical layer. There is no time-out event here because all outstanding frames are acknowledged before the timer expires. Note that although ACK 2 is lost, ACK 3 serves as both ACK 2 and ACK 3.

Figure 11.16 *Flow diagram for Example 11.6*



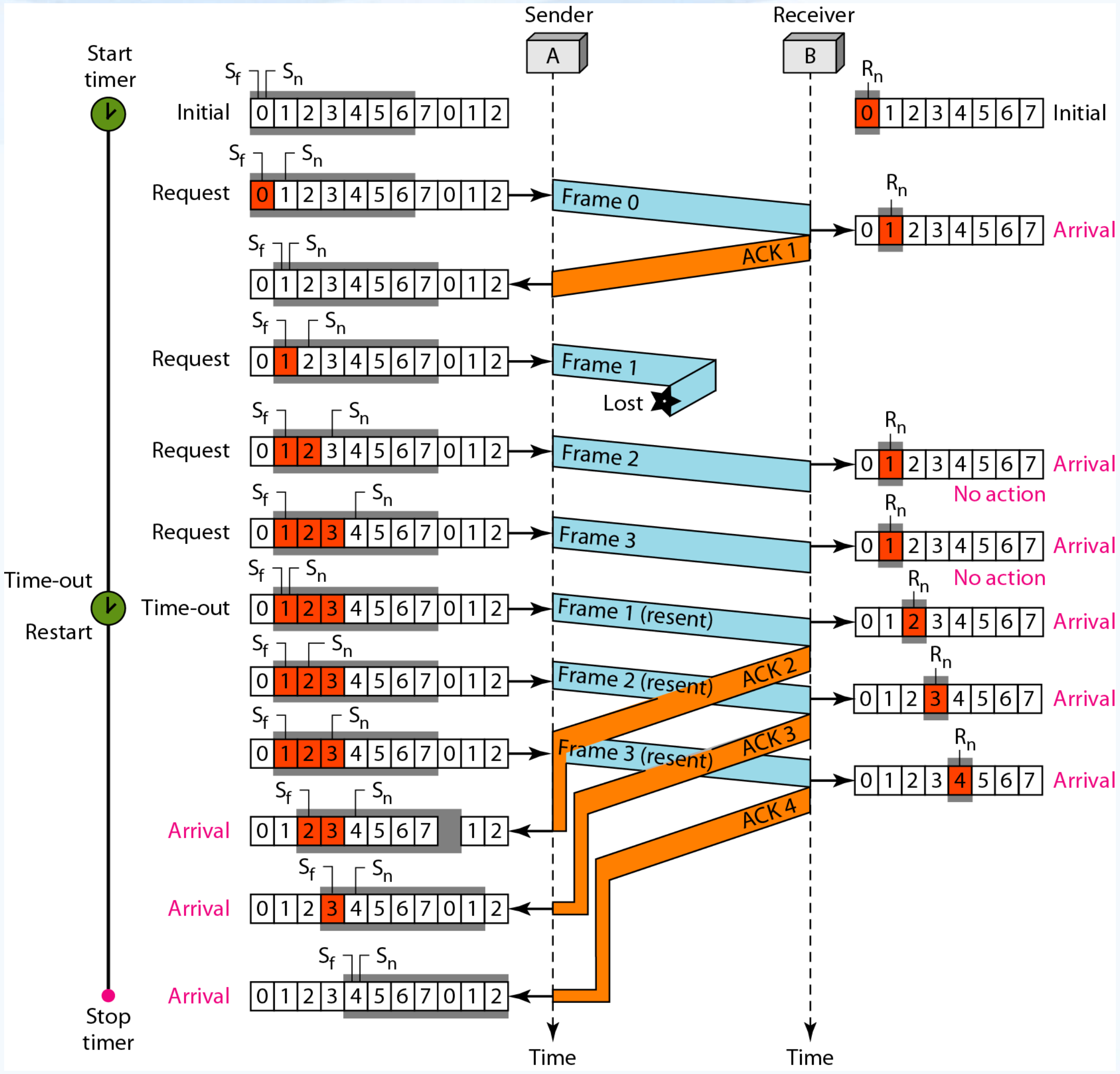
Example 11.7

Figure 11.17 shows what happens when a frame is lost. Frames 0, 1, 2, and 3 are sent. However, frame 1 is lost. The receiver receives frames 2 and 3, but they are discarded because they are received out of order. The sender receives no acknowledgment about frames 1, 2, or 3. Its timer finally expires. The sender sends all outstanding frames (1, 2, and 3) because it does not know what is wrong. Note that the resending of frames 1, 2, and 3 is the response to one single event. When the sender is responding to this event, it cannot accept the triggering of other events. This means that when ACK 2 arrives, the sender is still busy with sending frame 3.

Example 11.7 (continued)

The physical layer must wait until this event is completed and the data link layer goes back to its sleeping state. We have shown a vertical line to indicate the delay. It is the same story with ACK 3; but when ACK 3 arrives, the sender is busy responding to ACK 2. It happens again when ACK 4 arrives. Note that before the second timer expires, all outstanding frames have been sent and the timer is stopped.

Figure 11.17 *Flow diagram for Example 11.7*



Selective Repeat Automatic Repeat Request

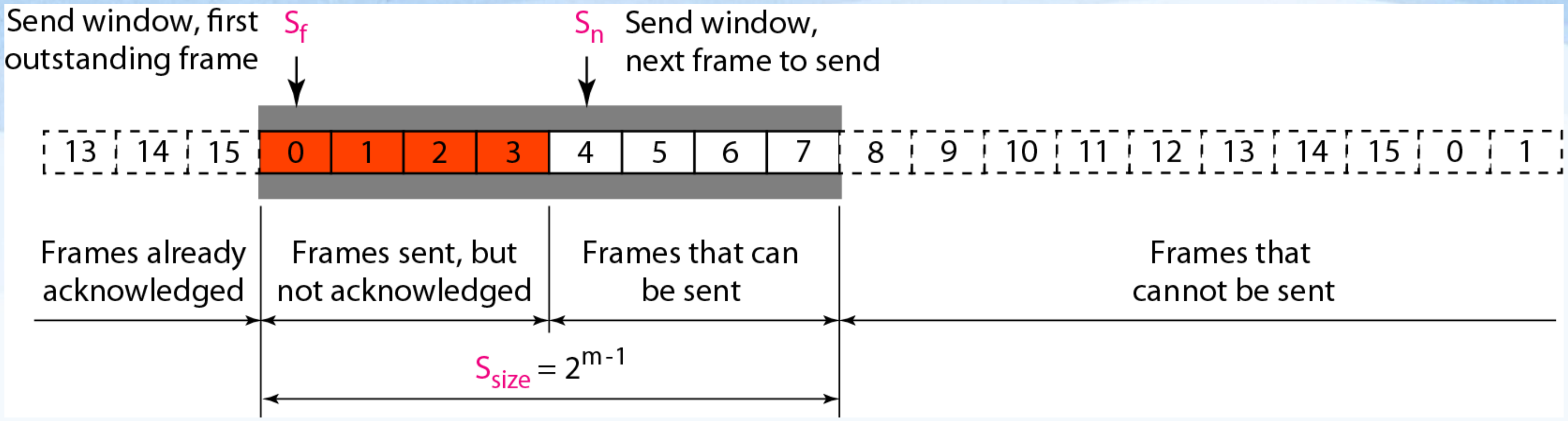
Go-Back-N ARQ simplifies the process at the receiver site. The receiver keeps track of only one variable, and there is no need to buffer out-of-order frames; they are simply discarded.
However, this protocol is very inefficient for a noisy link. In a noisy link a frame has a higher probability of damage, which means the resending of multiple frames. This resending uses up the bandwidth and slows down the transmission. For noisy links, there is another mechanism that does not resend N frames when just one frame is damaged; only the damaged frame is resent. This mechanism is called Selective Repeat ARQ.

Windows

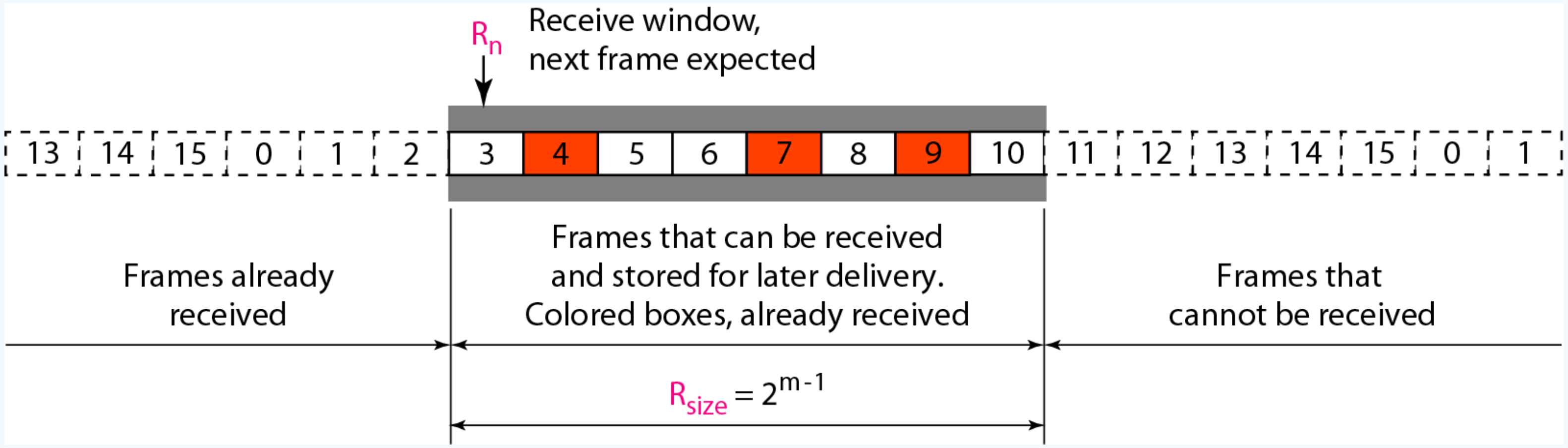
The Selective Repeat Protocol also uses two windows: a send window and a receive window. However, there are differences between the windows in this protocol and the ones in Go-Back-N. First, the size of the send window is much smaller; it is 2^m-1. Second, the receive window is the same size as the send window.

Send window for selective repeat ARQ

The send window maximum size can be 2^m- 1. For example, if m = 4, the sequence numbers go from 0 to 15, but the size of the window is just 8.The smaller window size means less efficiency in filling the pipe, but the fact that there are fewer duplicate frames can compensate for this.



Send window, first outstanding frame
S_f
S_n
Send window, next frame to send
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
Frames already acknowledged
Frames sent, but not acknowledged
Frames that can be sent
Frames that cannot be sent
$S_{size} = 2^{m-1}$



R_n
Receive window, next frame expected
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
Frames already received
Frames that can be received and stored for later delivery. Colored boxes, already received
Frames that cannot be received
$R_{size} = 2^{m-1}$

Piggybacking

Data frames flow in only one direction although control information such as ACK and NAK frames can travel in the other direction. In real life, data frames are normally flowing in both directions: from node A to node B and from node B to node A. This means that the control information also needs to flow in both directions.

When a frame is carrying data from A to B, it can also carry control information about arrived (or lost) frames from B; when a frame is carrying data from B to A, it can also carry control information about the arrived (or lost) frames from A. This is called Piggybacking. Piggybacking is used to improve the efficiency of the bidirectional protocols.

Example 11.8

This example is similar to Example 11.3 in which frame 1 is lost. We show how Selective Repeat behaves in this case. Figure 11.23 shows the situation. One main difference is the number of timers. Here, each frame sent or resent needs a timer, which means that the timers need to be numbered (0, 1, 2, and 3). The timer for frame 0 starts at the first request, but stops when the ACK for this frame arrives. The timer for frame 1 starts at the second request, restarts when a NAK arrives, and finally stops when the last ACK arrives. The other two timers start when the corresponding frames are sent and stop at the last arrival event.

Example 11.8 (continued)

At the receiver site we need to distinguish between the acceptance of a frame and its delivery to the network layer. At the second arrival, frame 2 arrives and is stored and marked, but it cannot be delivered because frame 1 is missing. At the next arrival, frame 3 arrives and is marked and stored, but still none of the frames can be delivered. Only at the last arrival, when finally a copy of frame 1 arrives, can frames 1, 2, and 3 be delivered to the network layer. There are two conditions for the delivery of frames to the network layer: First, a set of consecutive frames must have arrived. Second, the set starts from the beginning of the window.

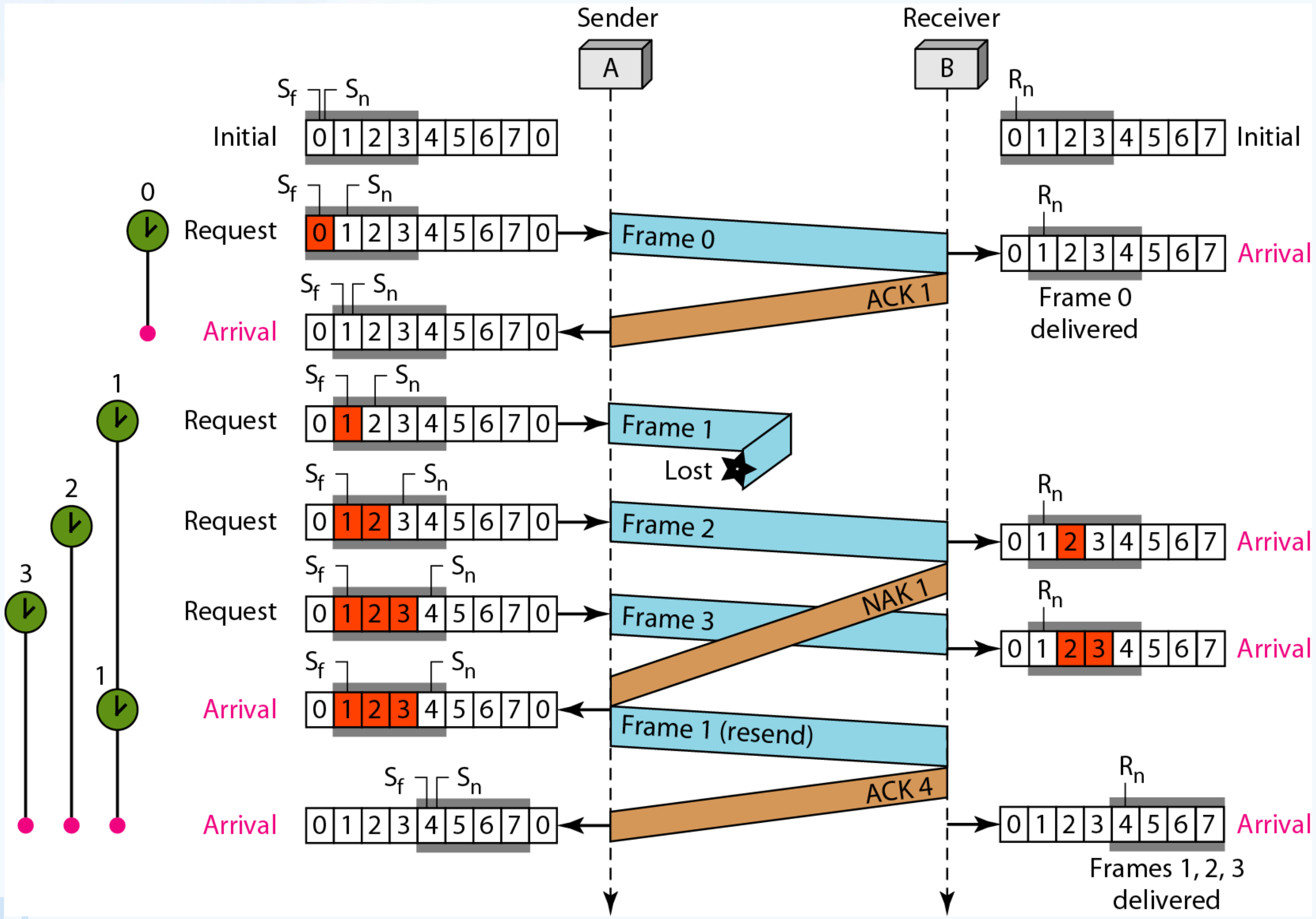
Example 11.8 (continued)

Another important point is that a NAK is sent after the second arrival, but not after the third, although both situations look the same. The reason is that the protocol does not want to crowd the network with unnecessary NAKs and unnecessary resent frames. The second NAK would still be NAK1 to inform the sender to resend frame 1 again; this has already been done. The first NAK sent is remembered (using the nakSent variable) and is not sent again until the frame slides. A NAK is sent once for each window position and defines the first slot in the window.

Example 11.8 (continued)

The next point is about the ACKs. Notice that only two ACKs are sent here. The first one acknowledges only the first frame; the second one acknowledges three frames. In Selective Repeat, ACKs are sent when data are delivered to the network layer. If the data belonging to n frames are delivered in one shot, only one ACK is sent for all of them.

Figure 11.23 *Flow diagram for Example 11.8*



Practice Problems

A bit string, 0111101111101111110, needs to be transmitted at the data link layer. What is the string actually transmitted after bit stuffing?

The output is 011110111110011111010.

A channel has a bit rate of 4 kbps and a propagation delay of 20 msec. For what range of frame sizes does stop-and-wait give an efficiency of at least 50%?

Efficiency will be 50% when the time required to transmit the frame equals the round-trip propagation delay. At a transmission rate of 4 bits/msec, 160 bits takes 40 msec. For frame sizes above 160 bits, stop-and-wait is reasonably efficient.

Consider an error-free 64-kbps satellite channel used to send 512-byte data frames in one direction, with very short acknowledgements coming back the other way. What is the maximum throughput for window sizes of 1, 7, 15, and 127? The earth-satellite propagation time is 270 msec.

The transmission starts at t=0.
At t=4096/64000 sec = 64 msec, the last bit is sent.
At t=334 msec, the last bit arrives at the satellite and the very short ACK is sent.
At t=604 msec, the ACK arrives at the earth.
The data rate here is 4096 bits in 604 msec, or about 6781 bps.
With a window size of 7 frames, transmission time is 448 msec for the full window, at which time the sender has to stop.
At 604 msec, the first ACK arrives and the cycle can start again.
Here we have 7 ×4096 =28,672 bits in 604 msec. The data rate is 47,470.2 bps.

Continuous transmission can only occur if the transmitter is still sending when the first ACK gets back at t=604 msec. In other words, if the window size is greater than 604 msec worth of transmission, it can run at full speed.
For a window size of 10 or greater this condition is met, so for any window size of 10 or greater (e.g., 15 or 127) the data rate is 64 kbps.

Consider a 50-kbps channel with a propagation delay between sender and receiver being 75 milliseconds. If the sender is trying to send a 1200-bit frame to the receiver then what is the optimal window size sender should have?

To calculate the optimal window size for the sender, we need to use the formula for the maximum window size:

W = (Bandwidth * Delay) / (Frame size * (1 - Error rate))

where:

- Bandwidth = 50 kbps (kilobits per second)
- Delay = 75 ms (milliseconds)
- Frame size = 1200 bits
- Error rate = 0 (assuming no errors in the transmission)

Substituting the given values, we get:
W = (50 * 10^3 * 75 * 10^-3) / (1200 * (1 - 0)) W = 3750 / 1200 W = 3.125

Since the window size must be an integer, we round up to the nearest integer, giving a final window size of 4.

Therefore, the optimal window size for the sender to use in this scenario is 4.