Chapter 10
Error Detection and Correction

Reference:

Data Communication and Networking, Behrouz A.Forouzan, McGraw Hill, 5th Edition, 2008

Note to Students : ppt is for revision purpose only. Answers in internals and exams should be written elaborately as given in the prescribed text book

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.
- 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.
- If the receiver receives an invalid codeword, this indicates that the data was corrupted during transmission.

Error Detection (1)

- **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.

- The sender creates codewords out of datawords by using a generator that applies the rules and procedures of encoding .

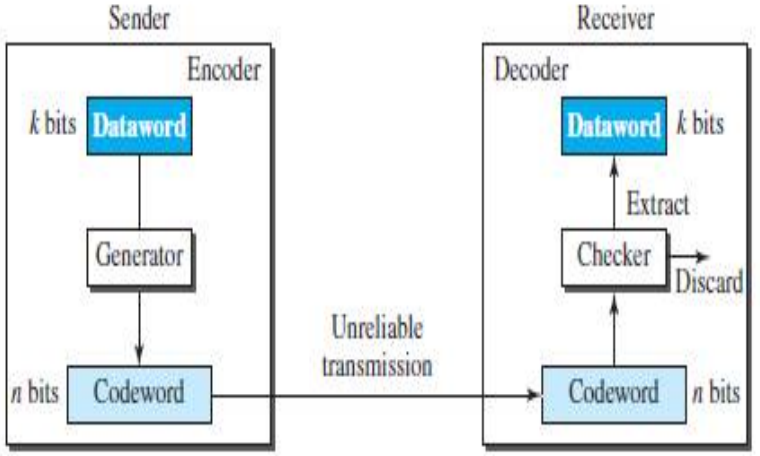
- Each codeword sent to the receiver may change during transmission.

- 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.

Figure 10.2 *Process of error detection in block coding*



Error Detection (2)

- **Example 10.1**
- Let us assume that k = 2 and n = 3. Table 10.1 shows the list of datawords and codewords

Table 10.1 *A code for error detection in Example 10.1*

Dataword	Codeword	Dataword	Codeword
00	000	10	101
01	011	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 (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.

Error Detection (3) –Hamming Distance

- **Hamming Distance**
- The Hamming distance between two words (of the same size) is the number of differences between the corresponding bits. It is represented d(x, y) where x and y are two words.
- 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.
- The Hamming distance can easily be found if we apply the XOR operation ($\oplus$) on the two words and count the number of 1s in the result. Its value can be greater than or equal to zero.

- **Example 10.2**
- 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).

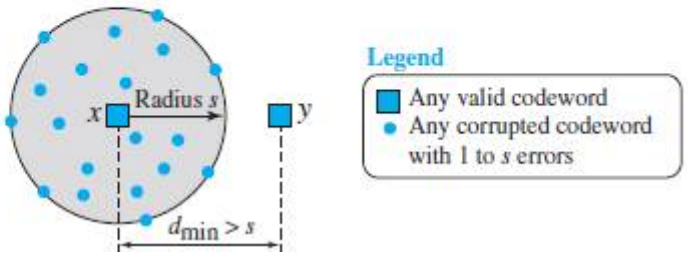
Error Detection (4) –Hamming Distance

- **Minimum Hamming Distance for Error Detection**
- In a set of codewords, the minimum Hamming distance is the smallest Hamming distance between all possible pairs of codewords.

- If our system is to detect up to s errors, the minimum distance between the valid codes must be (s + 1), so that the received codeword does not match a valid codeword.

- We can look at this criteria geometrically.
  - Let us assume that the sent codeword x is at the center of a circle with radius s.
  - All received codewords that are created by 0 to s errors are points inside the circle or on the perimeter of the circle.
  - All other valid codewords must be outside the circle, as shown in Figure 10.3.
  - This means that d_{min} must be an integer greater than s or $d_{min} = s + 1$.

Figure 10.3 *Geometric concept explaining d_{min} in error detection*



15. 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)

15.

a. d (10000, 00000) = **1**

b. d (10101, 10000) = **2**

c. d (1111, 1111) = **0**

d. d (000, 000) = **0**

16. Find the minimum Hamming distance for the following cases:
    a. Detection of two errors.
    b. Correction of two errors.
    c. Detection of 3 errors or correction of 2 errors.
    d. Detection of 6 errors or correction of 2 errors.

Note for correcting s errors, $d_{min} = 2t+1$

16.

a. For error detection $\rightarrow$ $d_{min} = s + 1 = 2 + 1 = 3$

b. For error correction $\rightarrow$ $d_{min} = 2t + 1 = 2 \times 2 + 1 = 5$

c.

For error section $\rightarrow$ $d_{min} = s + 1 = 3 + 1 = 4$
For error correction $\rightarrow$ $d_{min} = 2t + 1 = 2 \times 2 + 1 = 5$
Therefore d_{min} should be 5.

d.

For error detection $\rightarrow$ $d_{min} = s + 1 = 6 + 1 = 7$
For error correction $\rightarrow$ $d_{min} = 2t + 1 = 2 \times 2 + 1 = 5$
Therefore d_{min} should be 7.

Error Detection (5) – Linear Block Codes

- **Linear Block Codes**
- A linear block code is a code in which the exclusive OR (addition modulo-2) of two valid codewords creates another valid codeword.

- **Example 10.5**
- The code in Table 10.1 is a linear block code because the result of XORing any codeword with any other codeword is a valid codeword.

Table 10.1 *A code for error detection in Example 10.1*

Dataword	*Codeword*	*Dataword*	*Codeword*
00	000	10	101
01	011	11	110

- **Minimum Distance for Linear Block Codes**
- The minimum Hamming distance is the number of 1s in the nonzero valid codeword with the smallest number of 1s.

- **Example 10.6**
- In our first code (Table 10.1), the numbers of 1s in the nonzero codewords are 2, 2, and 2. So the minimum Hamming distance is $d_{min} = 2$.

18. Prove that the code represented by Table 10.8 is not a linear code. You need to find only one case that violates the linearity.

Table 10.8 *Table for Exercise 18*

Dataword	*Codeword*
00	00000
01	01011
10	10111
11	11111

18. We show that the exclusive-or of the second and the third code word

$$(01011) \oplus (10111) = 11100$$

is not in the code. The code is not linear.

Error Detection (6) – Parity Check Code

- **Parity-Check Code –** This 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.
- 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 10.1) is a parity-check code (k = 2 and n = 3). The code in Table 10.2 is also a parity-check code with k = 4 and n = 5.

Table 10.1 *A code for error detection in Example 10.1*

Dataword	Codeword	Dataword	Codeword
00	000	10	101
01	011	11	110

Table 10.2 *Simple parity-check code C(5, 4)*

Dataword	Codeword	Dataword	Codeword
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

Error Detection (7) – Parity Check Code

- The calculation is done in modular arithmetic .
- The encoder uses a generator that takes a copy of a 4-bit dataword (a0, a1, a2, and a3) and generates a parity bit r0.
- The dataword bits and the parity bit create the 5-bit codeword.
- The parity bit that is added makes the number of 1s in the codeword even.
- This is normally done by adding the 4 bits of the dataword (modulo-2); the result is the parity bit.
- In other words,
- If the number of 1s is even, the result is 0; if the number of 1s is odd, the result is 1. In both cases, the total number of 1s in the codeword is even.

$$r_0 = a_3 + a_2 + a_1 + a_0 \qquad \text{(modulo-2)}$$

- The sender sends the codeword, which may be corrupted during transmission. The receiver receives a 5-bit word.
- The checker at the receiver does the same thing as the generator in the sender with one exception: The addition is done over all 5 bits

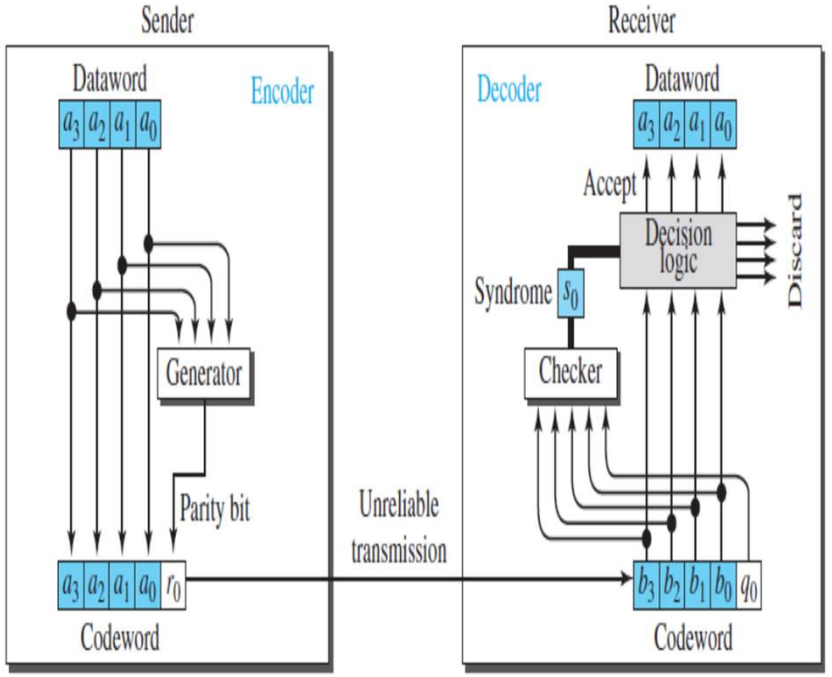
Error Detection (8) – Parity Check Code

- The result, which is called the syndrome, is just 1 bit. The syndrome is 0 when the number of 1s in the received codeword is even; otherwise, it is 1.

$$s_0 = b_3 + b_2 + b_1 + b_0 + q_0 \quad \text{(modulo-2)}$$

- The syndrome is passed to the decision logic analyzer. If the syndrome is 0, there is no detectable error in the received codeword; the data portion of the received codeword is accepted as the dataword; if the syndrome is 1, the data portion of the received codeword is discarded. The dataword is not created.

Figure 10.4 *Encoder and decoder for simple parity-check code*



Error Detection (9) – Parity Check Code

- **Example 10.7**
- 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.

29. Assuming even parity, find the parity bit for each of the following data units.
    a. 1001011
    b. 0001100
    c. 1000000
    d. 1110111

29. We need to add all bits modulo-2 (XORing). However, it is simpler to count the number of 1s and make them even by adding a 0 or a 1. We have shown the parity bit in the codeword in color and separate for emphasis.

	Dataword		Number of 1s		Parity	Codeword
a.	1001011	→	4 (even)	→	0	0 1001011
b.	0001100	→	2 (even)	→	0	0 0001100
c.	1000000	→	1 (odd)	→	1	1 1000000
d.	1110111	→	6 (even)	→	0	0 1110111

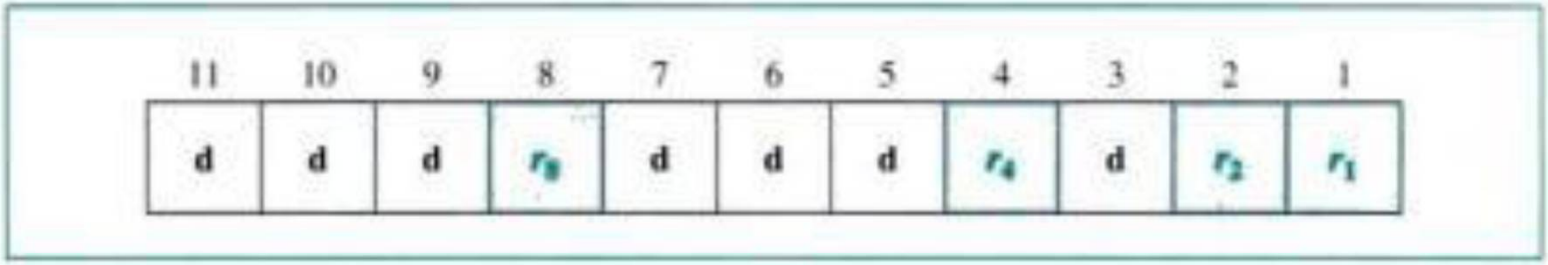
Hamming Code (1)

Hamming Code

Hamming provides a practical solution. The **Hamming code** can be applied to data units of any length and uses the relationship between data and redundancy bits discussed above. For example, a 7-bit ASCII code requires 4 redundancy bits that can be added to the end of the data unit or interspersed with the original data bits. In Figure 10.14, these bits are placed in positions 1, 2, 4, and 8 (the positions in an 11-bit sequence that are powers of 2). For clarity in the examples below, we refer to these bits as r_1, r_2, r_4, and r_8.

Hamming Code (2)

Figure 10.14 *Positions of redundancy bits in Hamming code*



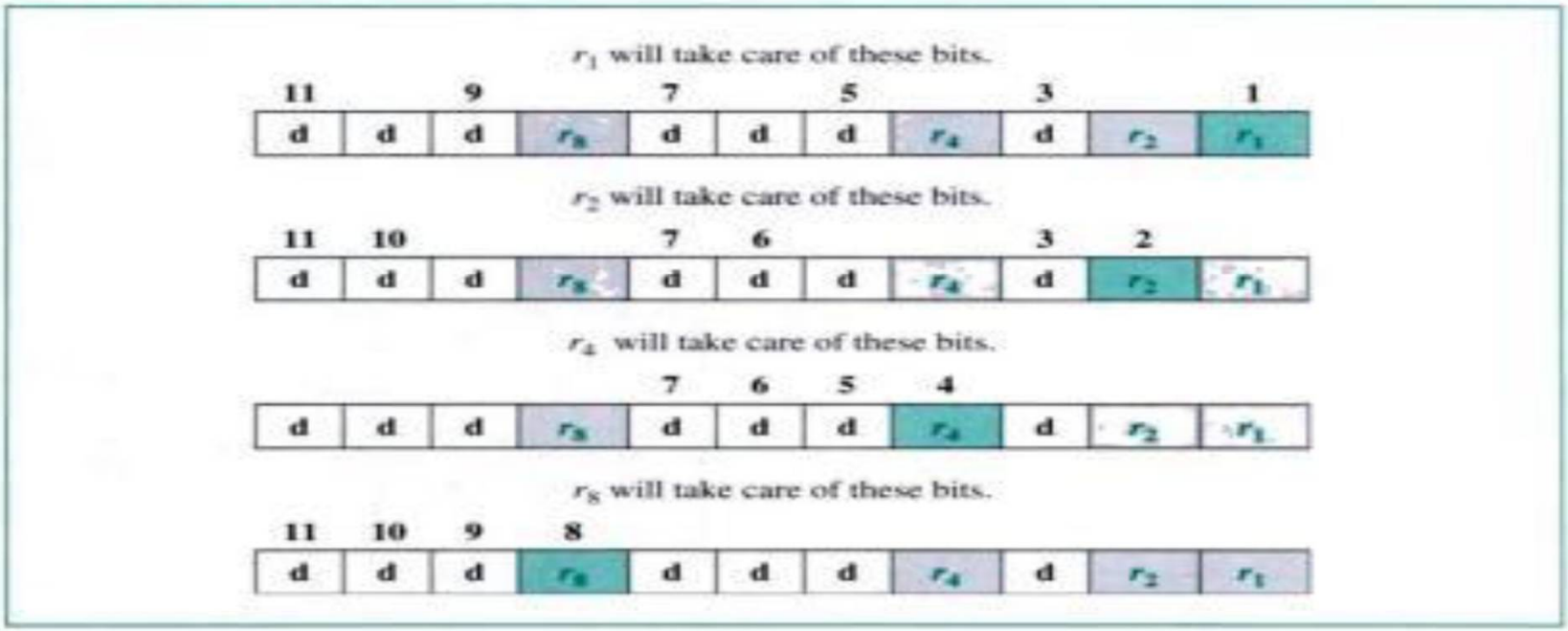
In the Hamming code, each r bit is the parity bit for one combination of data bits, as shown below:

r_1: bits 1, 3, 5, 7, 9, 11
r_2: bits 2, 3, 6, 7, 10, 11
r_4: bits 4, 5, 6, 7
r_8: bits 8, 9, 10, 11

Each data bit may be included in more than one calculation. In the sequences above, for example, each of the original data bits is included in at least two sets, while the r bits are included in only one (see Fig. 10.15).

Hamming Code (3)

Figure 10.15 *Redundancy bits calculation*

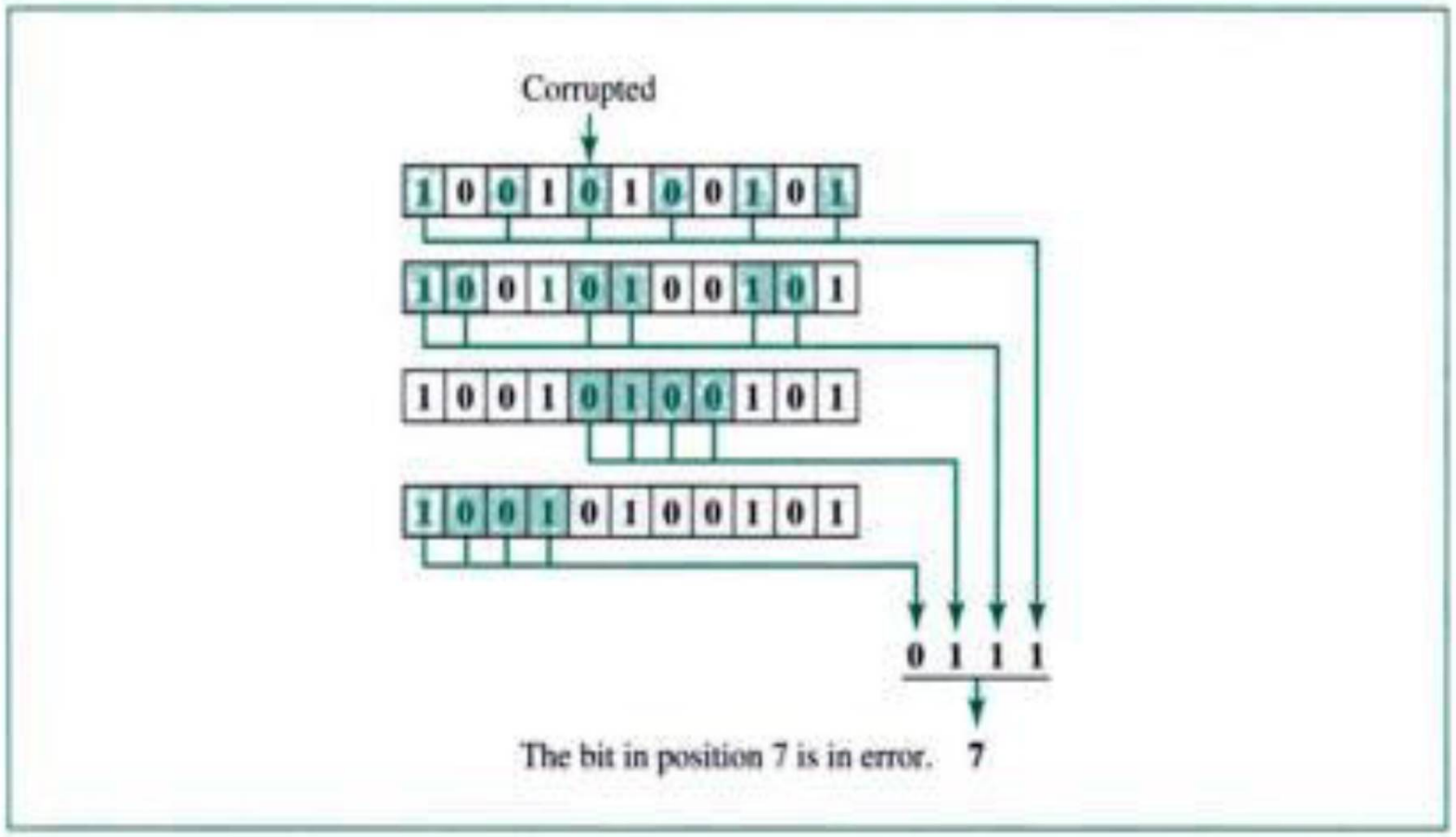


Calculating the *r* Values Figure 10.16 shows a Hamming code implementation for an ASCII character. In the first step, we place each bit of the original character in its appropriate position in the 11-bit unit. In the subsequent steps, we calculate the even parities for the various bit combinations. The parity value for each combination is the value of the corresponding *r* bit.

Error Detection and Correction Now imagine that by the time the above transmission is received, the number 7 bit has been changed from 1 to 0. The receiver takes the

Hamming Code (4)

Figure 10.17 *Error detection using Hamming code*



Hamming Code (5)

Once the bit is identified, the receiver can reverse its value and correct the error. The beauty of the technique is that it can easily be implemented in hardware and the code is corrected before the receiver knows about it.

Note: Hamming Code is not in your syllabus but is an important concept. You will be asked to implement this in next semester “computer networks laboratory” and instructions will not be repeated then.

Reference of this material is taken from:

CYCLIC CODES (1)

- 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:

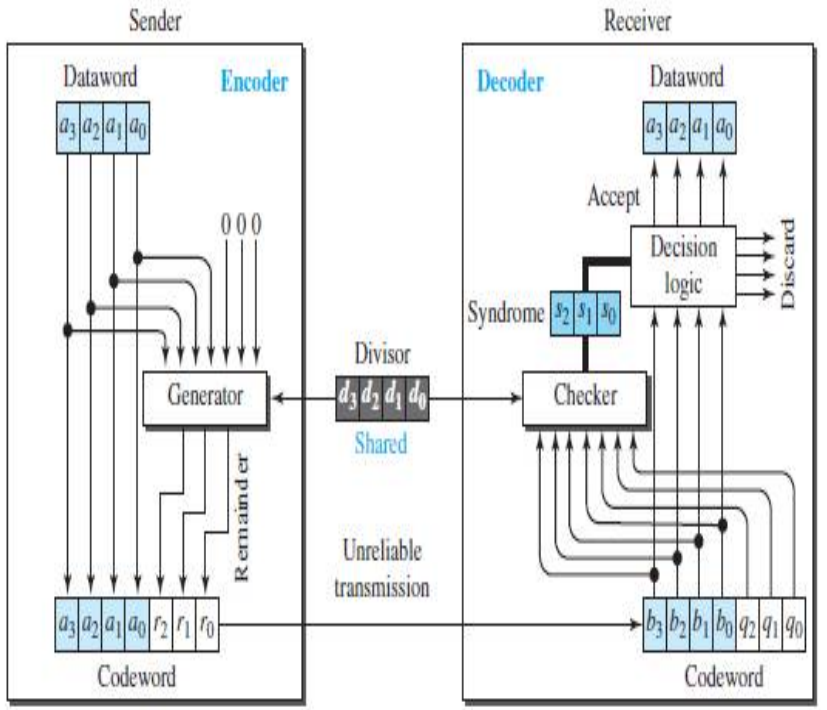
$$b_1 = a_0 \qquad b_2 = a_1 \qquad b_3 = a_2 \qquad b_4 = a_3 \qquad b_5 = a_4 \qquad b_6 = a_5 \qquad b_0 = a_6$$

- In the rightmost equation, the last bit of the first word is wrapped around and becomes the first bit of the second word.

CYCLIC CODES (2)– Cyclic Redundancy Check

- Figure 10.5 shows one possible design for the 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).
- The generator divides the augmented dataword by the divisor (modulo-2 division).

Figure 10.5 *CRC encoder and decoder*



CYCLIC CODES (3)– Cyclic Redundancy Check

- 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).

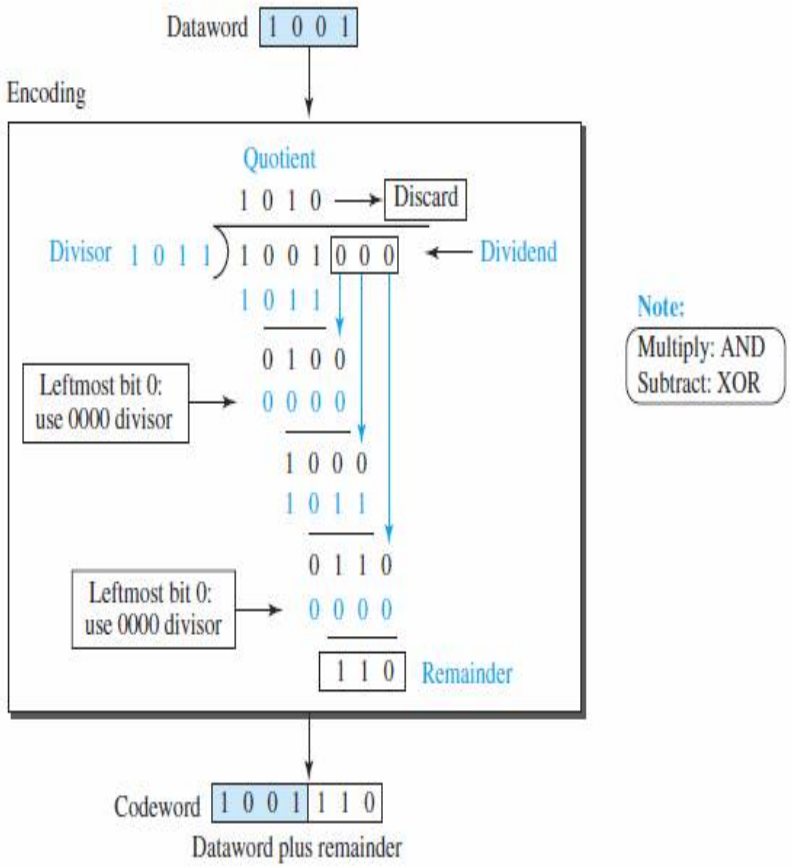
Table 10.3 *A CRC code with C(7, 4)*

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

CYCLIC CODES (4)– Cyclic Redundancy Check

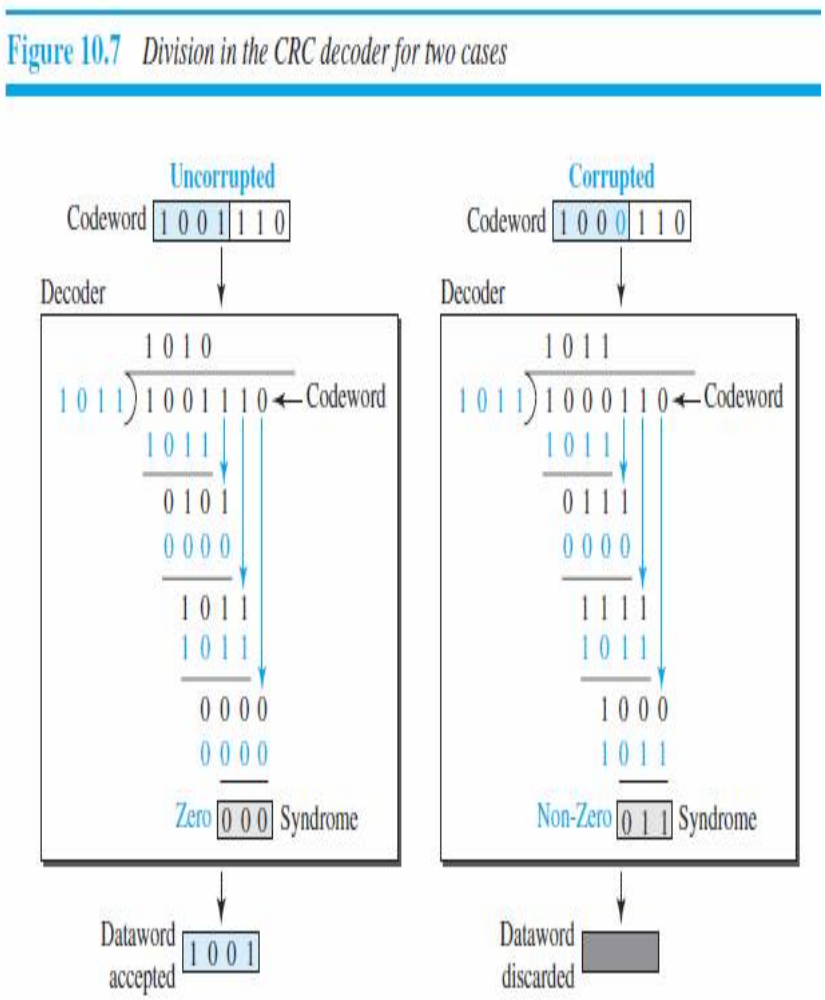
- **Encoder**
- The encoder takes a dataword and augments it with n – k number of 0s.
- It then divides the augmented dataword by the divisor
- The 3-bit remainder forms the check bits (r2, r1, and r0).
- They are appended to the dataword to create the codeword.

Figure 10.6 *Division in CRC encoder*



CYCLIC CODES (5)– Cyclic Redundancy Check

- **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.
- **Divisor**
  - Divisor chosen depends on the expectation we have from the code.

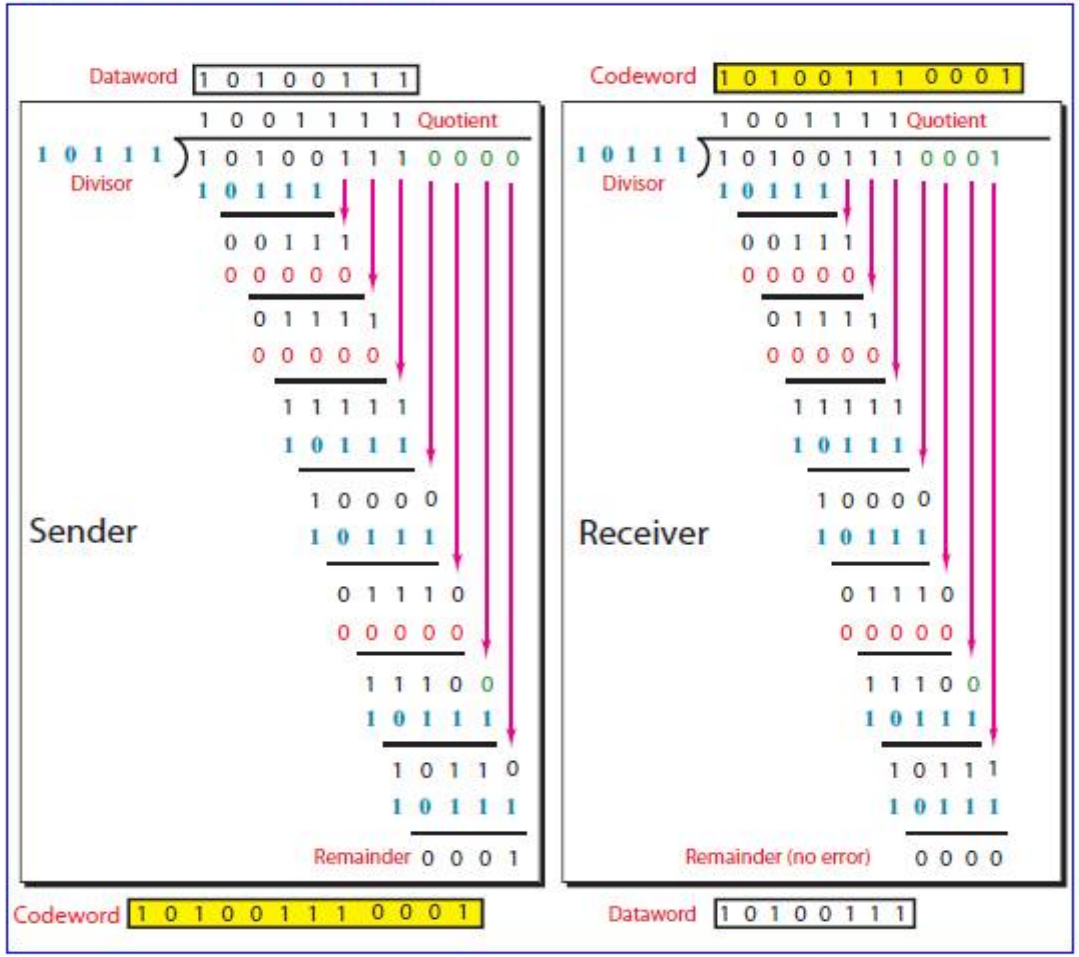


30. Given the dataword 10100111 and the divisor 10111,

a. Show the generation of the codeword at the sender site (using binary division).

b. Show the checking of the codeword at the receiver site (assume no error).

Figure 10.1 *Solution to Exercise 30*



CYCLIC CODES (6)– Polynomials

- **Polynomials**
- A better way to understand cyclic codes and how they can be analyzed is to represent them as polynomials.
- A pattern of 0s and 1s can be represented as a **polynomial with coefficients of 0 and** 1.
- The power of each term shows the position of the bit; the coefficient shows the value of the bit.
- Figure 10.8 shows a binary pattern and its polynomial representation.
- In Figure 10.8a we show how to translate a binary pattern into a polynomial; in Figure 10.8b we show how the polynomial can be shortened by removing all terms with zero coefficients and replacing x^1 *by* x *and* x^0 *by 1.*

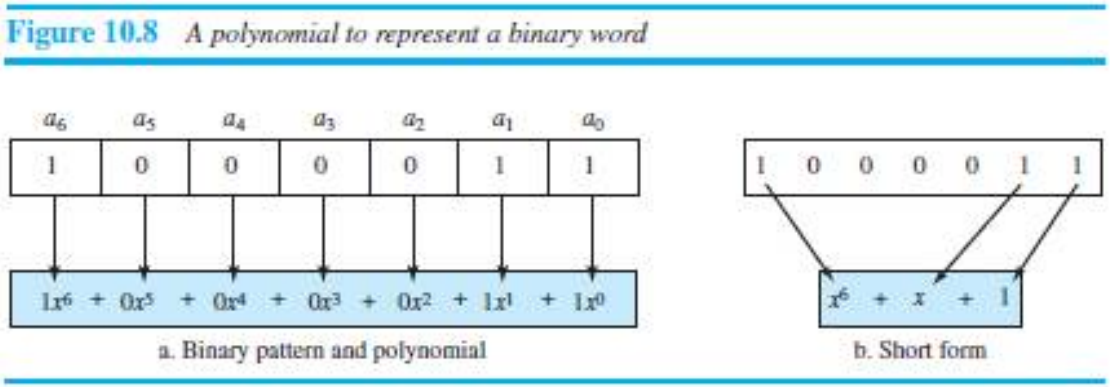


Figure 10.8 *A polynomial to represent a binary word*

CYCLIC CODES (7)– Polynomials

- ***Degree of a Polynomial***
  - The degree of a polynomial is the highest power in the polynomial.
  - For example, the degree of the polynomial $x^6 + x + 1$ is 6. Note that the degree of a polynomial is 1 less than the number of bits in the pattern. The bit pattern in this case has 7 bits.
- ***Adding and Subtracting Polynomials***
  - Adding and subtracting polynomials in mathematics are done by adding or subtracting the coefficients of terms with the same power.
  - In our case, the coefficients are only 0 and 1, and adding is in modulo-2.
  - This has two consequences. First, addition and subtraction are the same.
  - Second, adding or subtracting is done by combining terms and deleting pairs of identical terms.
  - For example, adding $x^5 + x^4 + x^2$ and $x^6 + x^4 + x^2$ gives just $x^6 + x^5$. The terms x^4 and x^2 are deleted.
- ***Multiplying or Dividing Terms***
  - In this arithmetic, multiplying a term by another term is very simple; we just add the powers. For example, $x^3 \times x^4$ is x^7.
  - For dividing, we just subtract the power of the second term from the power of the first. For example, x^5/x^2 is x^3.

CYCLIC CODES (8)– Polynomials

- ***Multiplying Two Polynomials***
  - Multiplying a polynomial by another is done term by term.
  - Each term of the first polynomial must be multiplied by all terms of the second. The result, of course, is then simplified, and pairs of equal terms are deleted. The following is an example:

$$(x^5+x^3+x^2+x)(x^2+x+1) = x^7+x^6+x^5+x^5+x^4+x^3+x^4+x^3+x^2+x^3+x^2+x$$
$$= x^7+x^6+x^3+x$$

- ***Dividing One Polynomial by Another***
  - Division of polynomials is conceptually the same as the binary division.
  - We divide the first term of the dividend by the first term of the divisor to get the first term of the quotient.
  - We multiply the term in the quotient by the divisor and subtract the result from the dividend.
  - We repeat the process until the dividend degree is less than the divisor degree.

- ***Shifting***
  - A binary pattern is often shifted a number of bits to the right or left.
  - Shifting to the left means adding extra 0s as rightmost bits; shifting to the right means deleting some rightmost bits.
  - Shifting to the left is accomplished by multiplying each term of the polynomial by x^m, where m is the number of shifted bits; shifting to the right is accomplished by dividing each term of the polynomial by x^m.
  - The following shows shifting to the left and to the right.

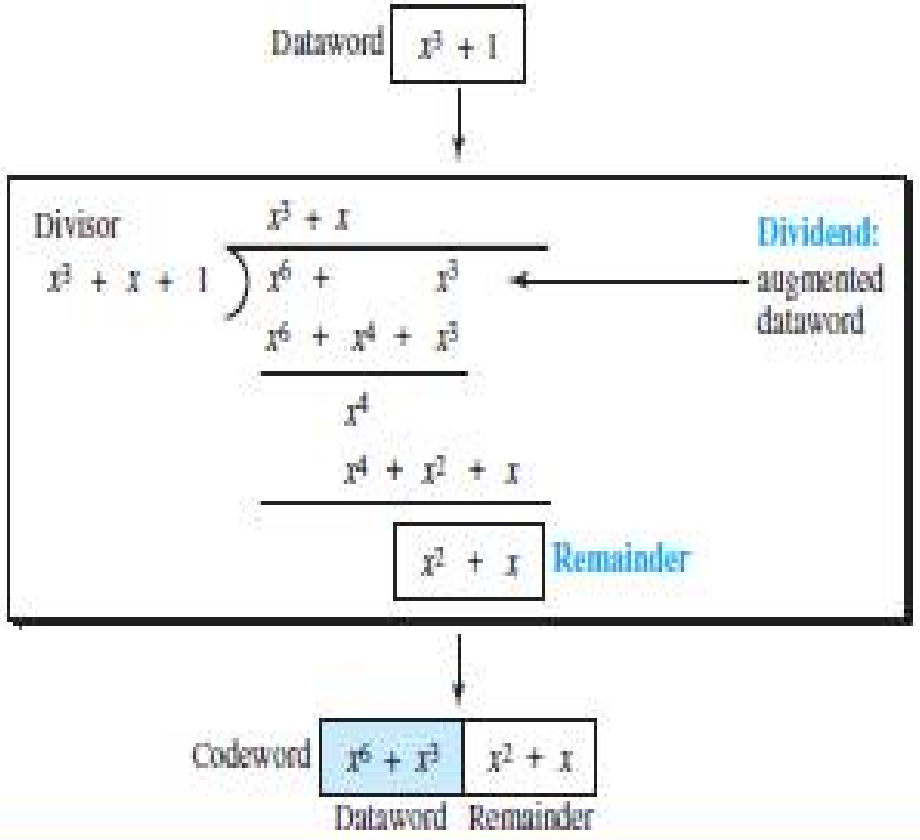
Shifting left 3 bits: 10011 becomes 10011000	x^4+x+1 becomes $x^7+x^4+x^3$
Shifting right 3 bits: 10011 becomes 10	x^4+x+1 becomes x

CYCLIC CODES (9)– Cyclic Code Encoder Using Polynomials

- Figure 10.9 is the polynomial version of Figure 10.6.
- The dataword 1001 is represented as $x^3 + 1$. The divisor 1011 is represented as $x^3 + x + 1$.
- To find the augmented dataword, we have left-shifted the dataword 3 bits (multiplying by x3). The result is $x^6 + x^3$.
- Division is straightforward. We divide the first term of the dividend, x^6, by the first term of the divisor, x^3. The first term of the quotient is then x^6/x^3, or x^3.
- Then we multiply x3 by the divisor and subtract the result from the dividend. The result is x^4, with a degree greater than the divisor's degree; we continue to divide until the degree of the remainder is less than the degree of the divisor.
- It can be seen that the polynomial representation can easily simplify the operation of division in this case, because the two steps involving all-0s divisors are not needed here.
- In a polynomial representation, the divisor is normally referred to as the ***generator polynomial t(x).***

CYCLIC CODES (10)– Cyclic Code Encoder Using Polynomials

Figure 10.9 *CRC division using polynomials*

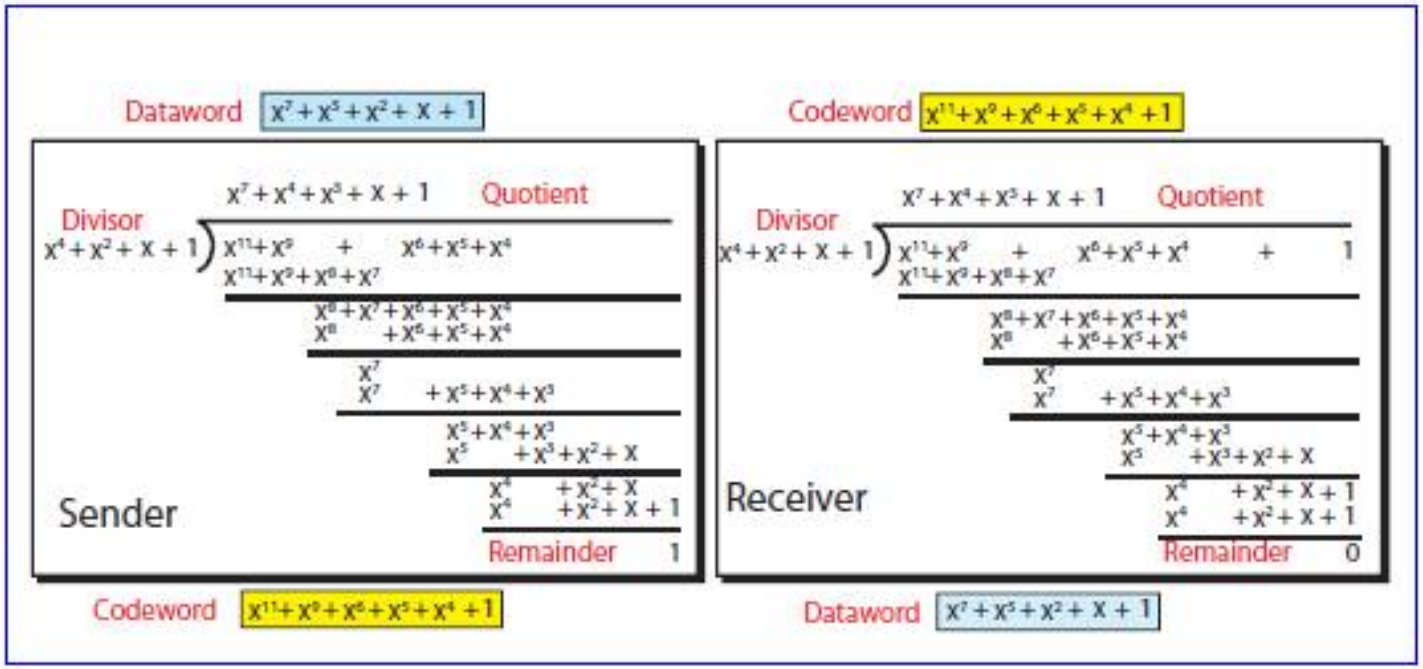


30. Given the dataword 10100111 and the divisor 10111,

a. Show the generation of the codeword at the sender site (using binary division).

h. Show the checking of the codeword at the receiver site (assume no error).

Figure 10.2 *Solution to Exercise 31*



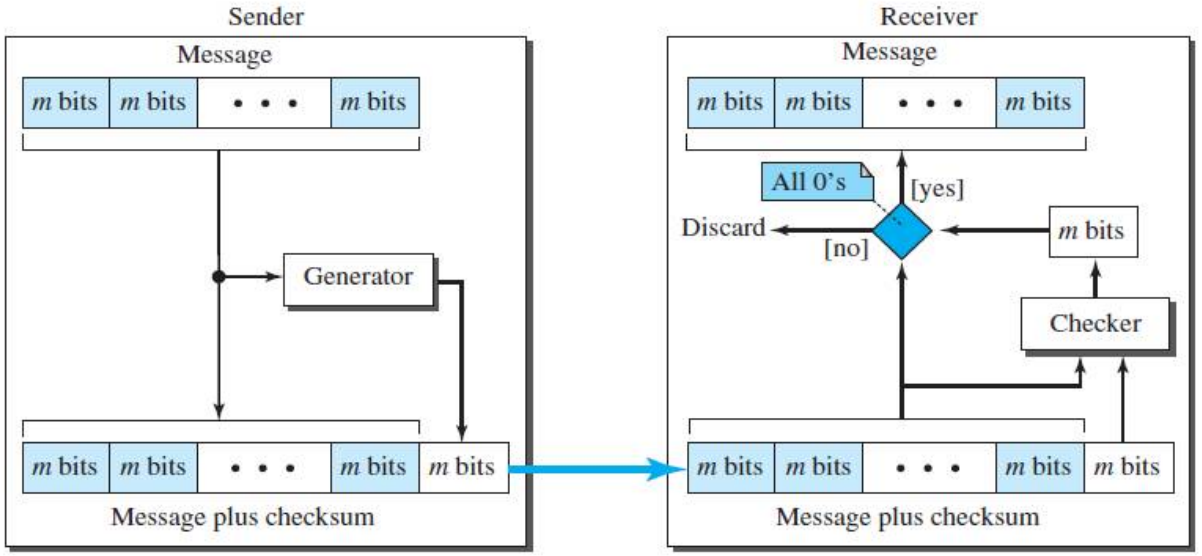
7. In CRC, show the relationship between the following entities (size means the number of bits):
   a. The size of the dataword and the size of the codeword
   b. The size of the divisor and the remainder
   c. The degree of the polynomial generator and the size of the divisor
   d. The degree of the polynomial generator and the size of the remainder

a. The only relationship between the size of the codeword and dataword is the one based on the definition: **n = k + r**., where *n* is the size of the codeword, *k* is the size of the dataword, and *r* is the size of the remainder.
b. The *remainder* is always *one bit smaller* than the *divisor*.
c. The *degree* of the generator polynomial is *one less than* the size of the *divisor*. For example, the CRC-32 generator (with the polynomial of degree 32) uses a 33-bit divisor.
d. The *degree* of the generator polynomial is the *same as* the size of the remainder (length of checkbits). For example, CRC-32 (with the polynomial of degree 32) creates a remainder of 32 bits.

CHECKSUM (1)

- Checksum is an error-detecting technique that can be applied to a message of any length.
- In the Internet, the checksum technique is mostly used at the network and transport layer rather than the data-link layer.
- At the source, the message is first divided into m-bit units.
- The generator then creates an extra m-bit unit called the checksum, which is sent with the message.
- At the destination, the checker creates a new checksum from the combination of the message and sent checksum.
- If the new checksum is all 0s, the message is accepted; otherwise, the message is discarded (Figure 10.15).

Figure 10.15 *Checksum*

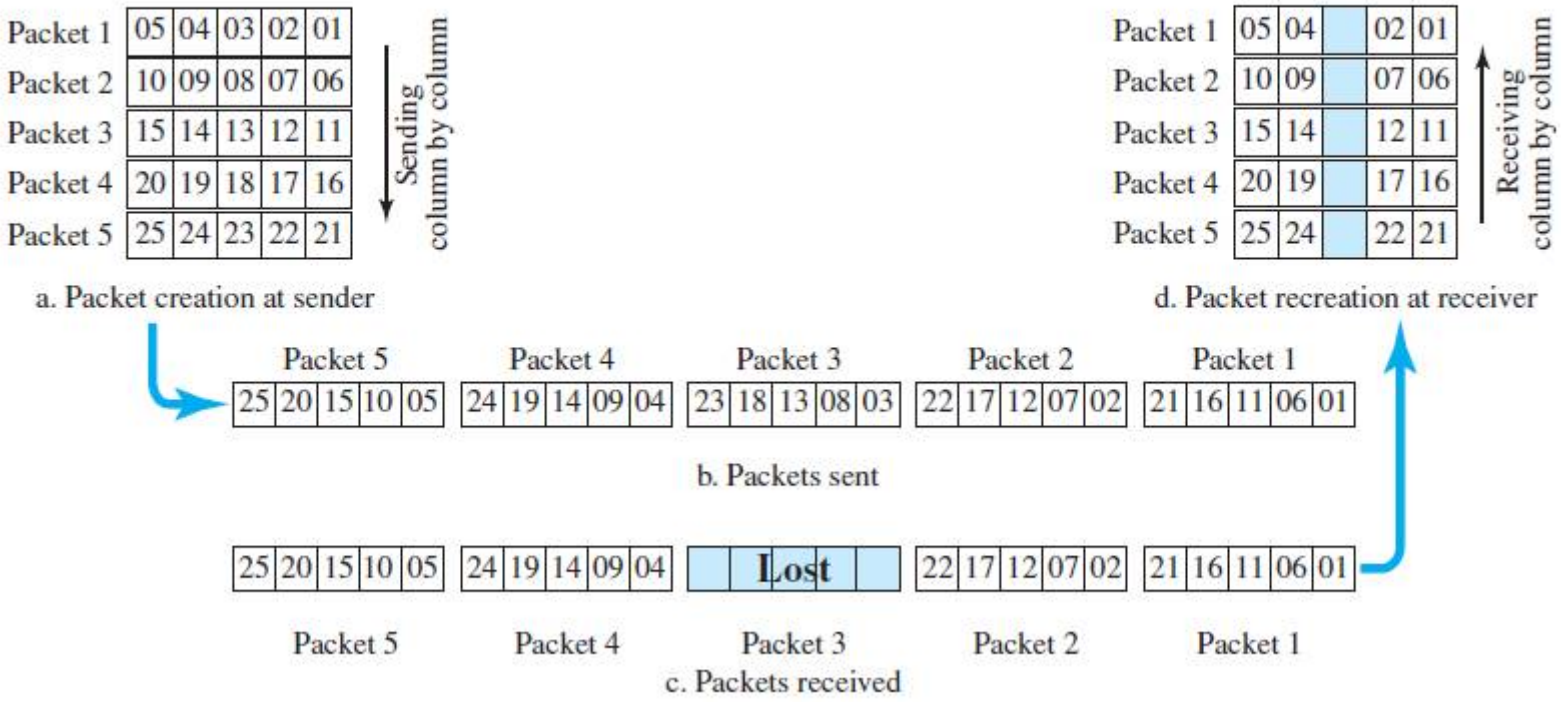


FORWARD ERROR CORRECTION (1) - Chunk Interleaving

- Retransmission of corrupted and lost packets is not useful for real-time multimedia transmission because it creates an unacceptable delay in reproducing: we need to wait until the lost or corrupted packet is resent.
- We need to correct the error or reproduce the packet immediately.
- Several schemes have been designed and used in this case that are collectively referred to as **forward error correction (FEC) techniques.**
- **Chunk Interleaving:**
  - Another way to achieve FEC in multimedia is to allow some small chunks to be missing at the receiver.
  - We cannot afford to let all the chunks belonging to the same packet be missing; however, we can afford to let one chunk be missing in each packet.
  - Figure 10.21 shows that we can divide each packet into 5 chunks (normally the number is much larger).
  - We can then create data chunk by chunk (horizontally), but combine the chunks into packets vertically.
  - In this case, each packet sent carries a chunk from several original packets.
  - If the packet is lost, we miss only one chunk in each packet, which is normally acceptable in multimedia communication.

FORWARD ERROR CORRECTION (2) - Chunk Interleaving

Figure 10.21 *Interleaving*



END of Chapter 10

Data Link Control (DLC)

Framing

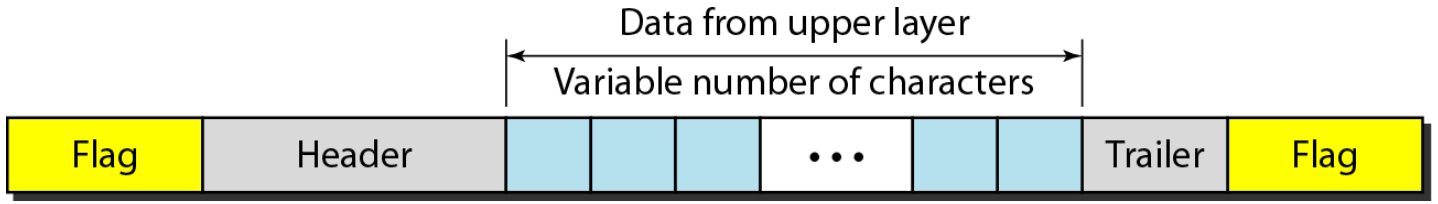
- *Framing in the data-link layer separates a message from one source to a destination* by adding a sender address and a destination address
  - The destination address defines where the packet is to go
  - The sender address helps the recipient acknowledge the receipt
- Why the whole message not packed in one frame?
  - 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?
    - Fixed-Size Framing
    - Variable-Size Framing

Framing ...

- Fixed-Size Framing
  - no need for defining the boundaries of the frames
  - the size itself can be used as a delimiter
  - An example of this type of framing is the ATM WAN, which uses frames of fixed size called *cells*
    - ATM: Asynchronous Transfer Mode(connection oriented, high-speed network technology that is used in both LAN and WAN over optical fiber and operates upto gigabit speed)
- Variable-Size Framing -
  - need a way to define the end of one frame and the beginning of the next
  - Historically, two approaches were used for this purpose:
    - a character-oriented *(or byte-oriented)* approach
    - a bit-oriented approach
  - prevalent in local-area networks

Character-Oriented Framing

- Data to be carried are 8-bit characters from a coding system such as ASCII
- The header (also multiples of 8 bits) normally carries
  - the source and destination addresses
  - other control information
  - the trailer, which carries error detection redundant 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



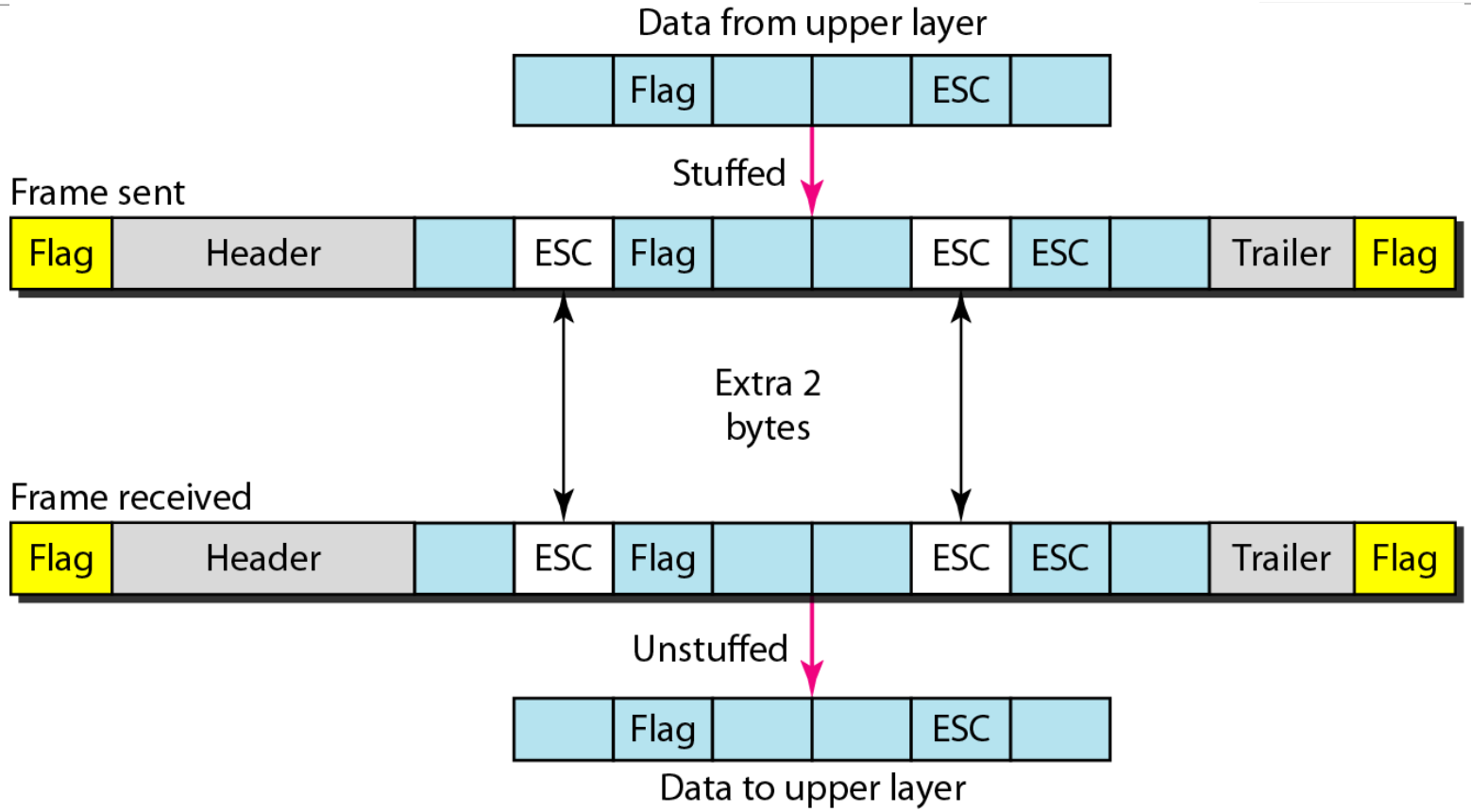
Problem with flag

- Text communication: flag could be selected to be any character not used for text communication
- Sending other types of information such as graphs, audio, and video: any character used for the flag could also be part of the information
- If flag is a part of the information, the receiver, when it encounters this pattern in the middle of the data, thinks it has reached the end of the frame

Solution to Flag Problem

- Add a byte-stuffing strategy 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
  - 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
- 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

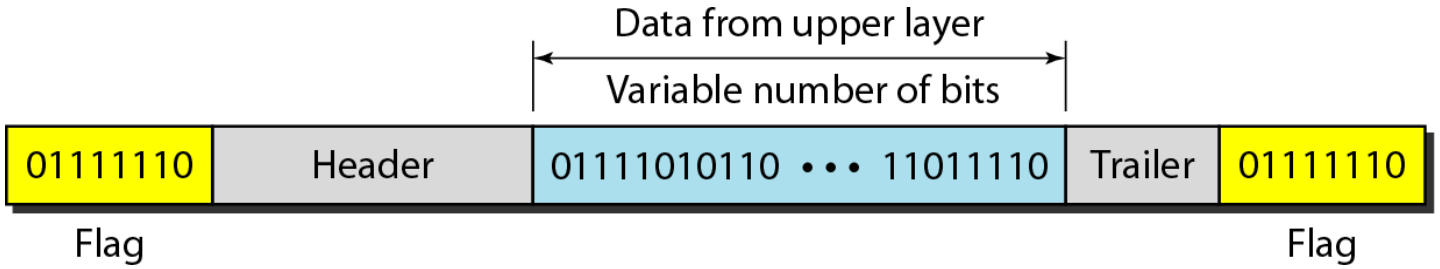
Byte stuffing and unstuffing



- The universal coding systems in use today, such as Unicode, have 16-bit and 32-bit characters that conflict with 8-bit characters
  - bit-oriented protocols

Bit-Oriented Framing

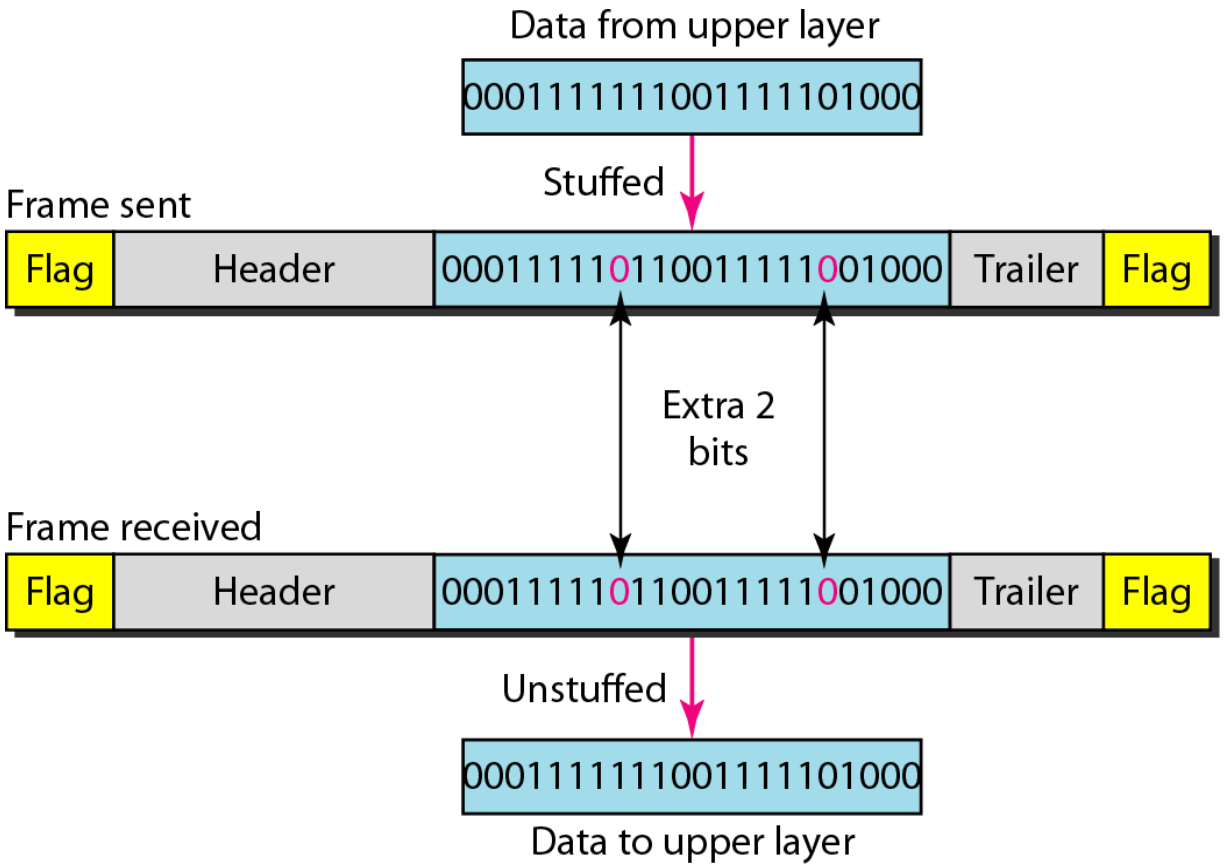
- Data section of a frame is a sequence of bits to be interpreted by the upper layer as text, graphic, audio, video, and so on
- In addition to headers (and possible trailers), need a delimiter to separate one frame from the other
  - Most protocols use a special 8-bit pattern flag, 01111110, as the delimiter to define the beginning and the end of the frame



Bit-Oriented Framing

- Bit stuffing is the process of adding one extra 0 whenever five consecutive 1s follow a 0 in the data, so that the receiver does not mistake the pattern 0111110 for a flag
- 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

Bit-Oriented Framing



Example 1

- Byte-stuff the following frame payload in which
  - E is the escape byte
  - F is the flag byte
  - D is a data byte
  - other than an escape or a flag character.

F D E E D D E F D D E E E E D E F D F

- Unstuff the following frame payload in which
  - E is the escape byte
  - F is the flag byte
  - D is a data byte
  - other than an escape or a flag character.

E D F D D F E D D D

Example 2

- Bit-stuff the following frame payload:

  000111111100111111010001111111111110000111

  0001111101100111111001000111111011111101000011 1

- Unstuff the following frame payload:

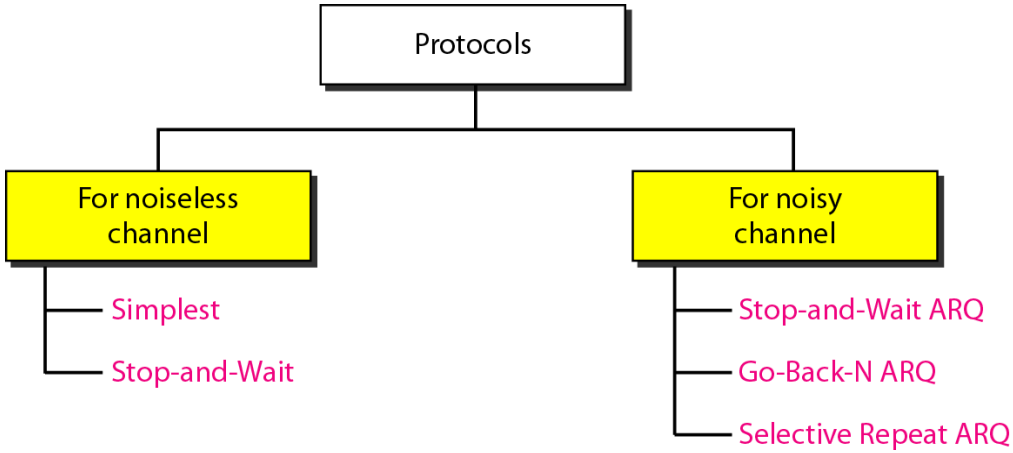
  000111110000011111011101001110111110000111

  1

  000111110000011111011101001110111110000011 11
  000111110000011111111010011101111100001111

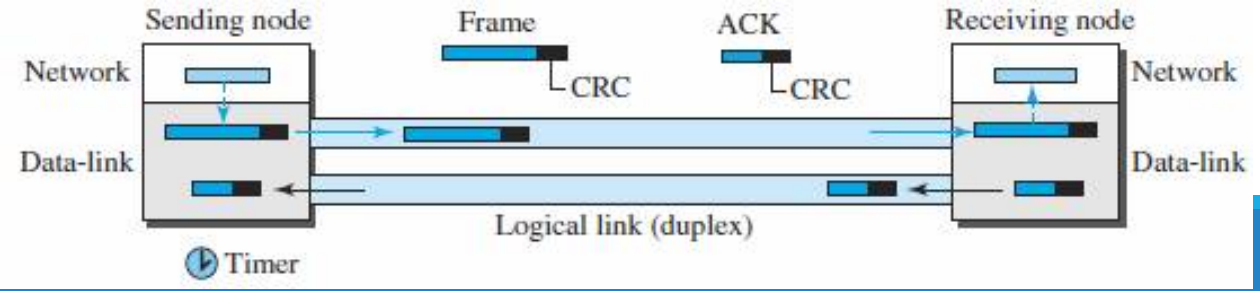
Data-link Layer Protocols

- Data link layer protocols can **combine framing, flow control, and error control** to achieve the delivery of data from one node to another
- The protocols are normally implemented in **software** by using one of the common programming languages
- four protocols have been defined for the data-link layer to deal with flow and error control:
  - Simple
  - Stop-and-Wait
  - Go-Back-N
  - Selective-Repeat

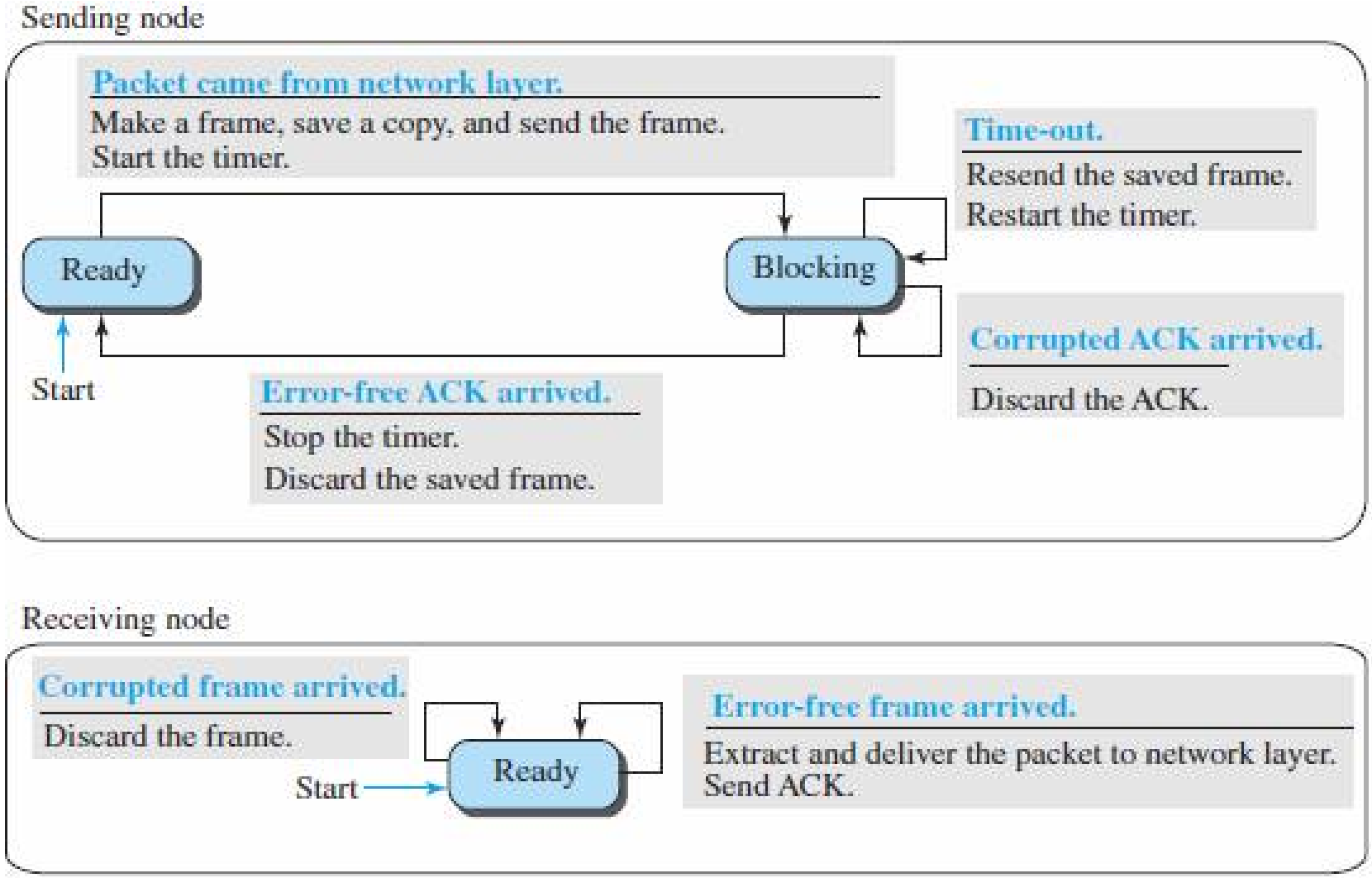


Stop-and-Wait Protocol

- Uses both flow and error control
- Sender sends one frame at a time and waits for an acknowledgment before sending the next one
- To detect corrupted frames, CRC is added to each data frame
- When a frame arrives at the receiver site, if its **CRC is incorrect**, the frame is corrupted and silently **discarded**
- The silence of the receiver is a signal for the sender that a frame was either corrupted or lost.
  - Every time the sender sends a frame, it starts a timer
  - If an acknowledgment arrives before the timer expires, the timer is stopped and the sender sends the next frame (if it has one to send)
  - If the timer expires, the sender resends the previous frame, assuming that the frame was either lost or corrupted
    - This means that the sender needs to keep a copy of the frame until its acknowledgment arrives
  - When the corresponding acknowledgment arrives, the sender discards the copy and sends the next frame if it is ready



Stop-and-Wait Protocol FSM



Stop-and-Wait Protocol FSM

Sender States

- The sender is initially in the ready state, but it can move between the ready and blocking state
- **Ready State:** only waiting for a packet from the network layer
  - If a packet comes from the network layer, the sender creates a frame, saves a copy of the frame, starts the only timer and sends the frame
  - The sender then moves to the blocking state.
- **Blocking State:** three events can occur:
  - If a time-out occurs, the sender resends the saved copy of the frame and restarts the timer
  - If a corrupted ACK arrives, it is discarded
  - If an error-free ACK arrives, the sender stops the timer and discards the saved copy of the frame
    - It then moves to the ready state.

Stop-and-Wait Protocol FSM

Receiver States

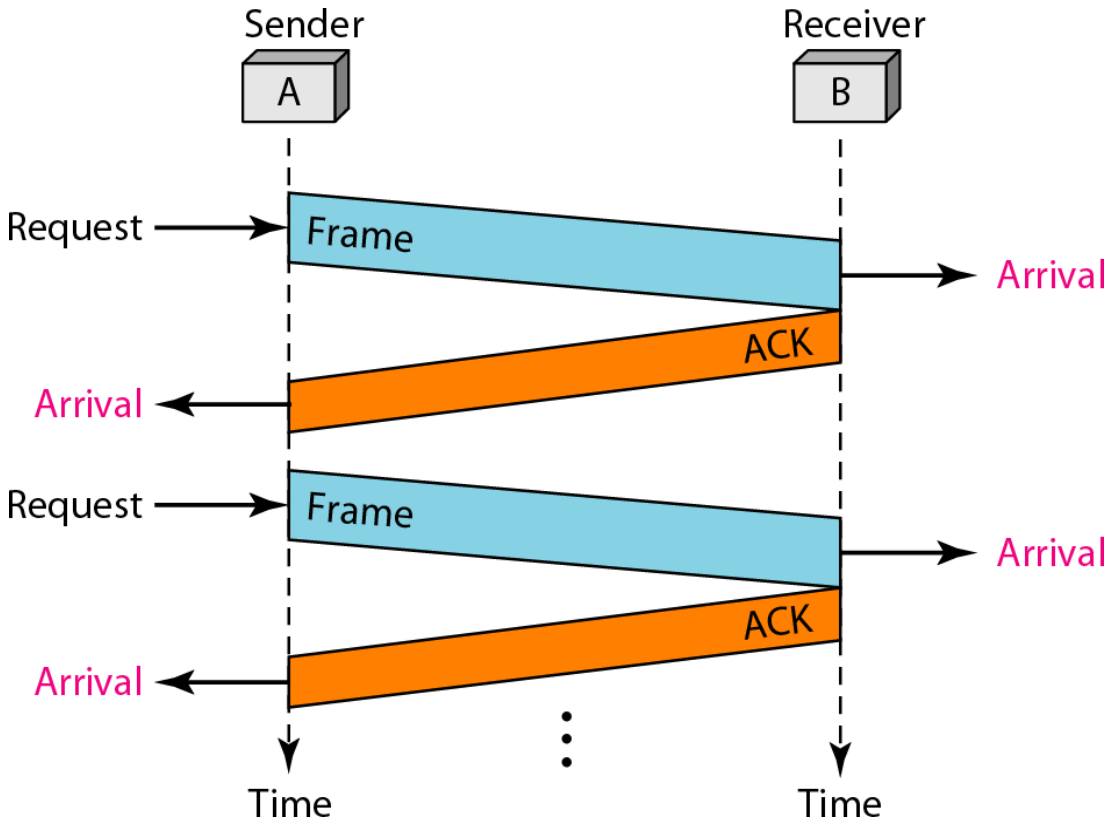
- The receiver is always in the ready state. Two events may occur:
  - If an error-free frame arrives, the message in the frame is delivered to the network layer and an ACK is sent
  - If a corrupted frame arrives, the frame is discarded

- **Cases of Operations:**

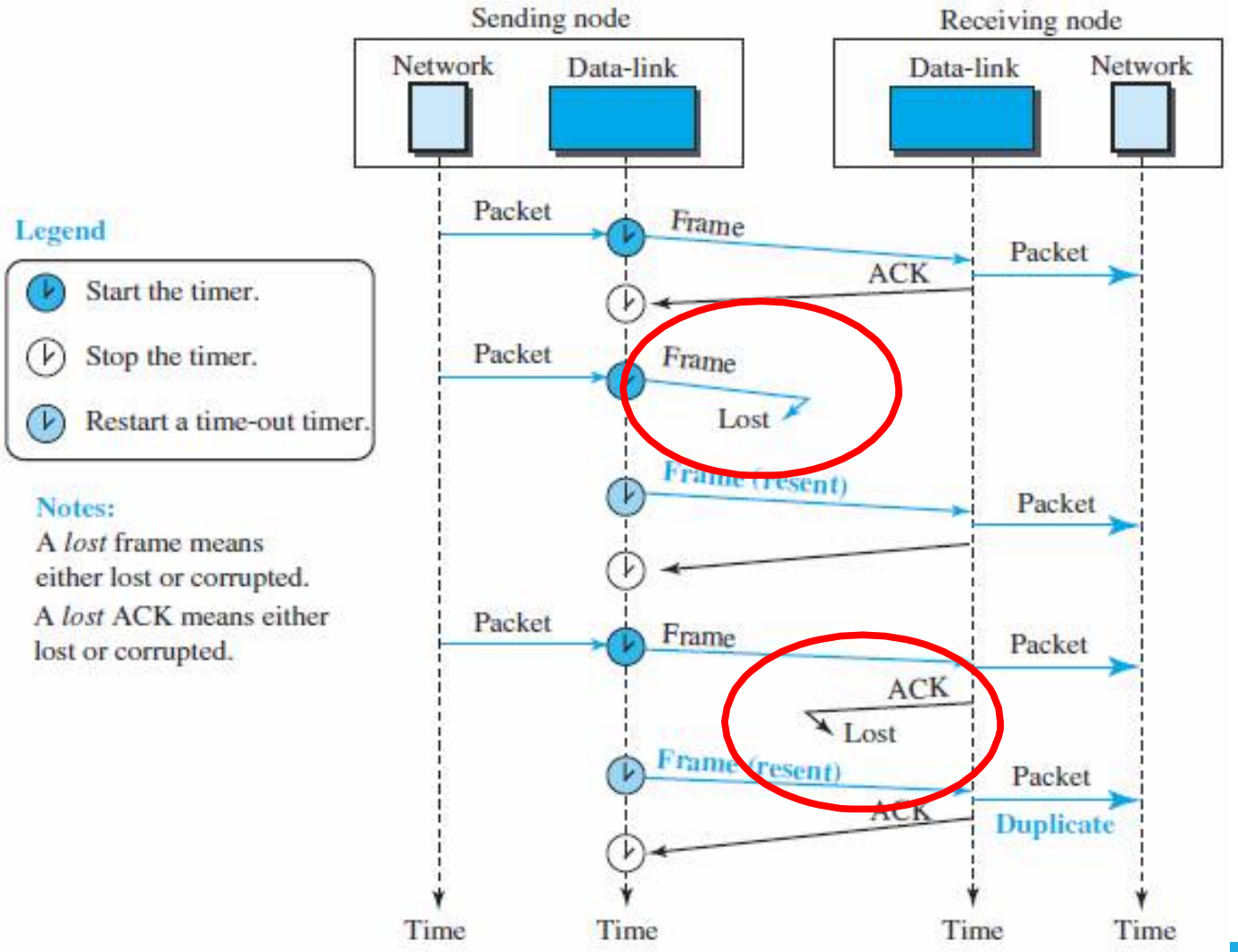
1. Normal operation
2. The frame is lost
3. The Acknowledgment (ACK) is lost
4. The ACK is delayed

Stop-and-Wait Protocol Example -1

- The sender sends one frame and waits for feedback from the receiver
- When the ACK arrives, the sender sends the next frame



Stop-and-Wait Protocol Example -2



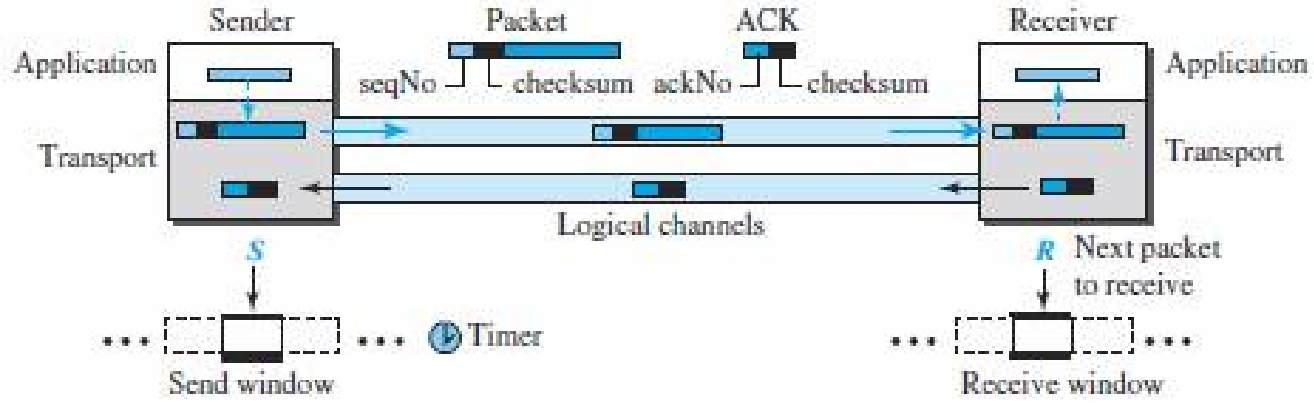
Stop-and-Wait Protocol Example -2

- The first frame is sent and acknowledged
- The second frame is sent, but lost
  - After time-out, it is resent
- The third frame is sent and acknowledged, but the acknowledgment is lost
  - The frame is resent
  - Problem: The network layer at the receiver site receives **two copies** of the third packet, which is not right
    - Solution: use **sequence numbers and acknowledgment numbers**

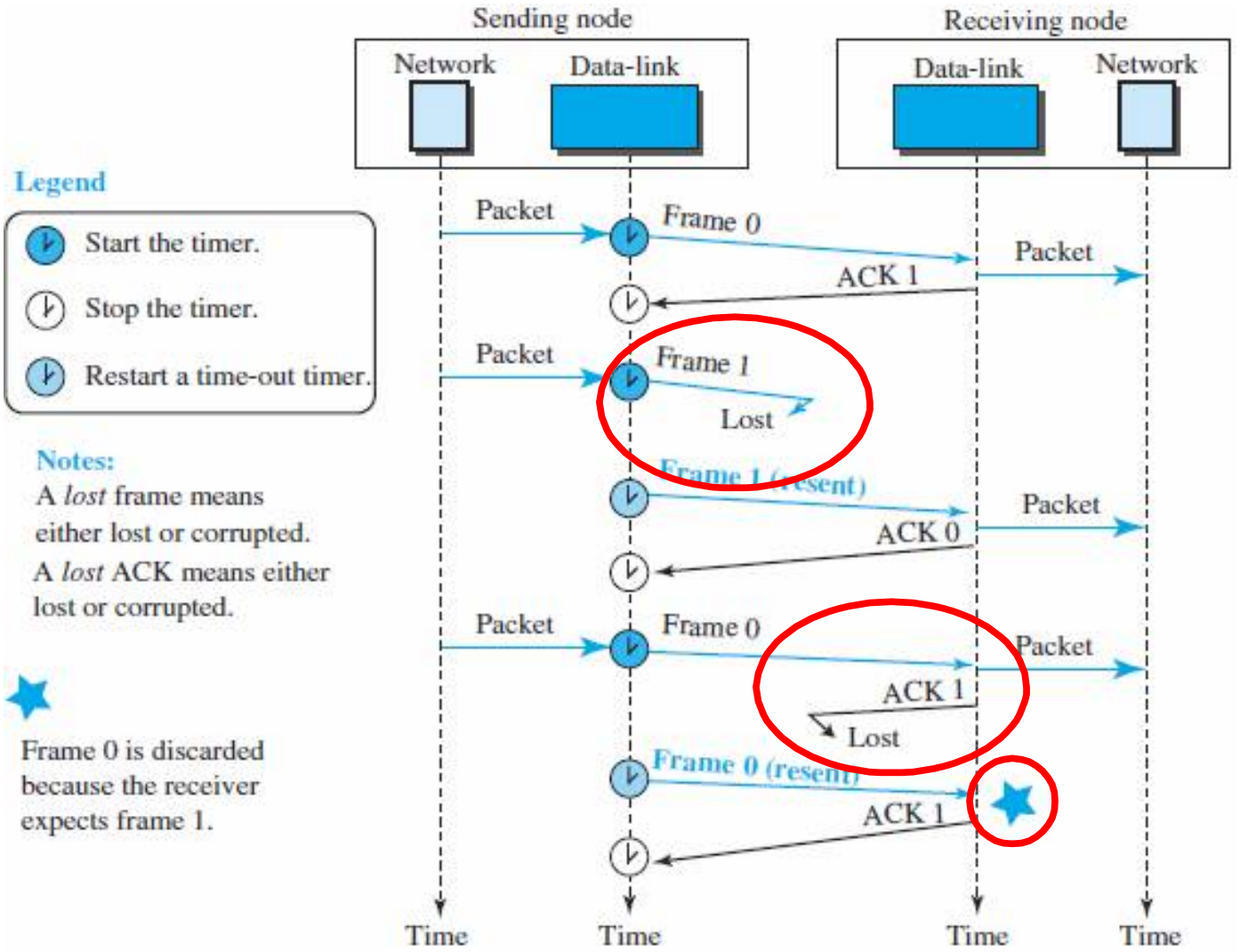
Stop-and-Wait Protocol

Sequence and Acknowledgment Numbers

- Duplicate packets need to be avoided
- Add sequence numbers to the data frames and acknowledgment numbers to the ACK frames
- The acknowledgment number always announces, in modulo-2 arithmetic, the sequence number of the next packet expected
  - Sequence numbers are 0, 1, 0, 1, 0, 1, . . . ; the acknowledgment numbers can also be 1, 0, 1, 0, 1, 0, ...
    - the sequence numbers start with 0, the acknowledgment numbers start with 1.
    - An acknowledgment number always defines the sequence number of the next frame to receive



Stop-and-Wait Protocol Example -3

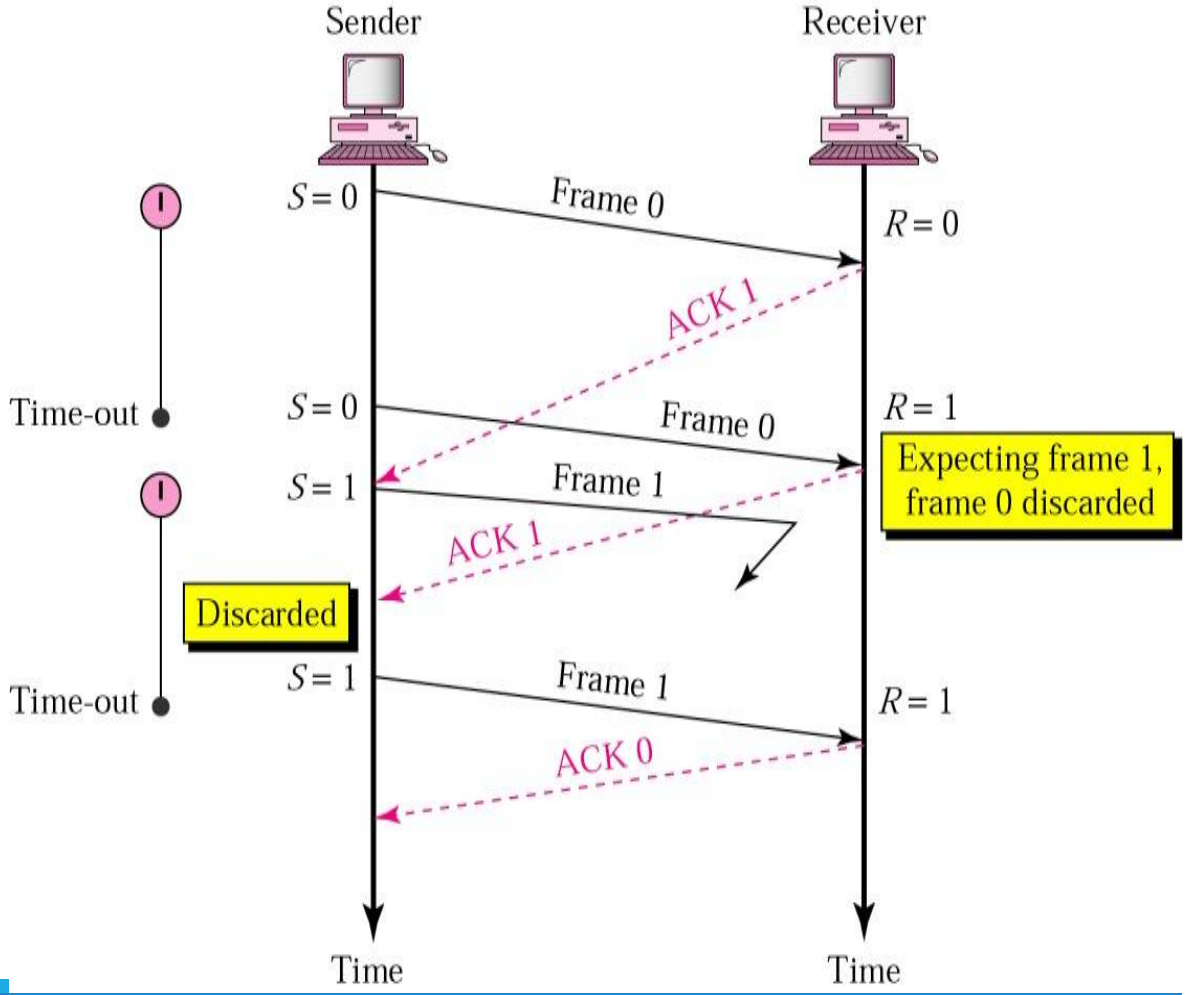


Stop-and-Wait Protocol Example -3

- Adding sequence numbers and acknowledgment numbers can prevent duplicates
  - The first frame is sent and acknowledged
  - The second frame is sent, but lost
    - After time-out, it is resent
  - The third frame is sent and acknowledged, but the acknowledgment is lost
    - The frame is resent

Stop-and-Wait Protocol Example -4

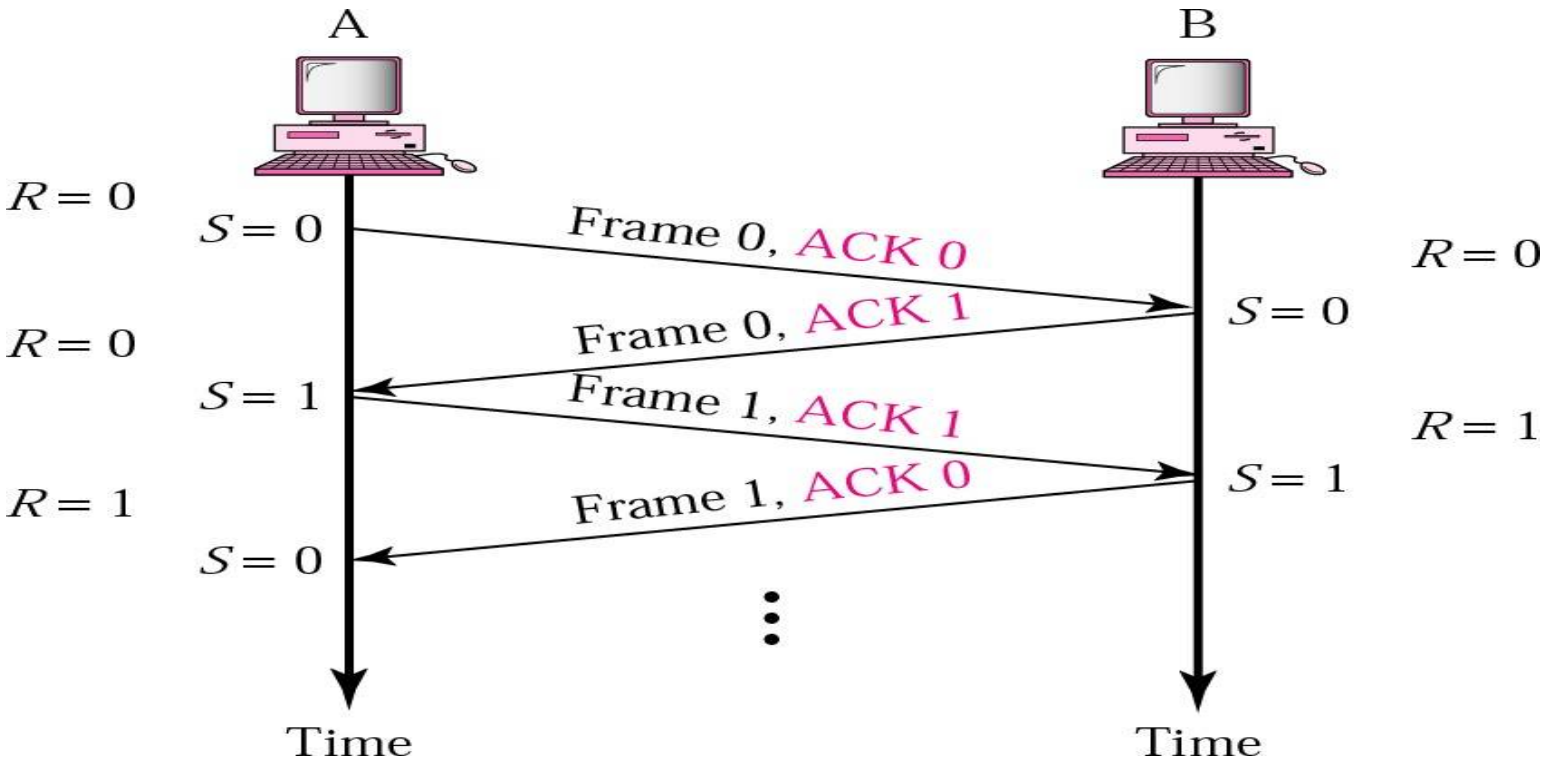
- **Delayed ACK and lost frame**



Piggybacking

- Simple/stop-and-wait protocols are designed
  - unidirectional communication, in which data is flowing only in one direction although the acknowledgment may travel in the other direction
- To allow data to flow in both directions to make the communication more efficient, the data in one direction is piggybacked with the acknowledgment in the other direction
  - When node A is sending data to node B, Node A also acknowledges the data received from node B
- Because piggybacking makes communication at the datalink layer **more complicated**, it is not a common practice

Flow diagram using piggybacking



Stop-and Wait - Limitations

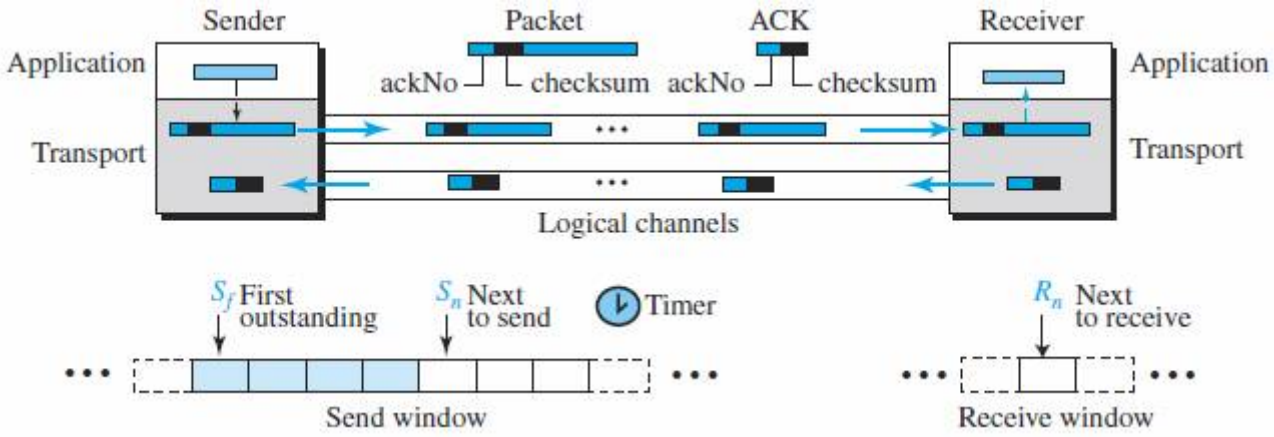
- After each frame sent, the host must wait for an ACK
  - ❖ inefficient use of bandwidth
- To improve efficiency ACK should be sent after multiple frames
  - Pipelining: A task is begun before the previous task has ended
  - ❖ There is no pipelining in stop and wait ARQ because we need to wait for a frame to reach the destination and be acknowledged before the next frame can be sent
  - ❖ Pipelining improves the efficiency of the transmission
  - Alternatives: Sliding Window protocol
  - ✓ Go-back-N ARQ
  - ✓ Selective Repeat ARQ

Sliding window protocols

- Sliding window protocols improve the efficiency
- Multiple frames should be in transition while waiting for ACK - Let more than one frame to be outstanding.
- Outstanding frames: frames sent but not acknowledged
- Send up to W frames and keep a copy of these frames (outstanding) until the ACKs arrive
- This procedures requires additional feature to be added : sliding window

Go-Back-*N* Protocol (GBN)

- Idea of Stop-and-Wait Protocol- how to add flow control to its predecessor
  - Error correction in Stop-and-Wait ARQ is done by keeping a copy of the sent frame and retransmitting of the frame when the timer expires
- Go-Back-N send several packets before receiving ACKs but the receiver can only buffer one packet
- Sender keep a copy of sent packets until ACKs arrive



The **sequence numbers** are modulo 2^m, where m is the size of the sequence number field in bits

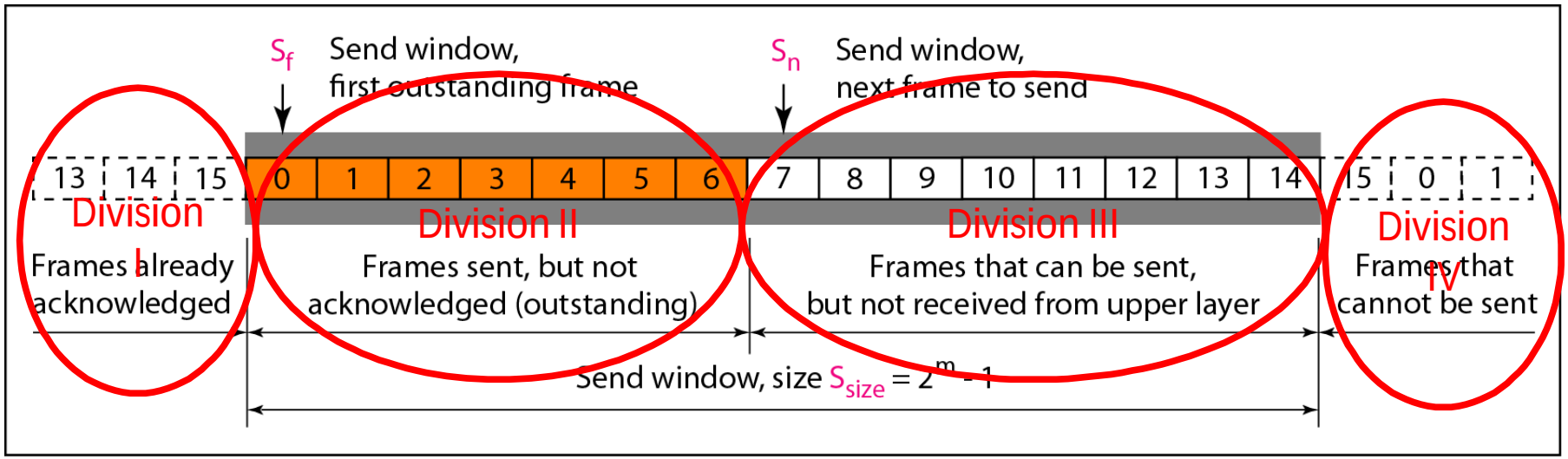
The **acknowledgment number** is **cumulative** and defines the sequence number of the next packet expected to arrive

GBN Windows

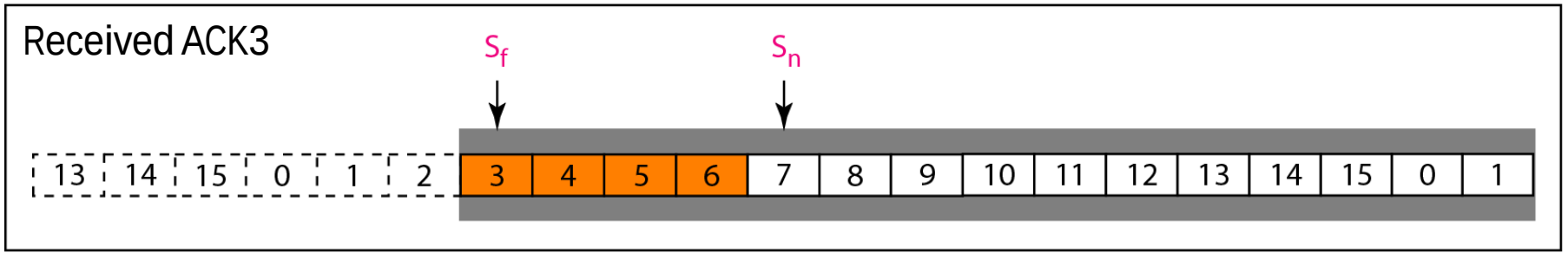
- The send window is an abstract concept defining an imaginary box of size $2^m - 1$ with three variables: S_f, S_n, and S_{size}
  - Covers the sequence numbers of the data packets that can be in transit or can be sent
  - The send window can slide one or more slots when a valid acknowledgment arrives
    - Ack number ahould be greater than or equal to S_f and less than S_n arrives
- The receive window is an abstract concept defining an imaginary box of size 1 with one single variable R_n
  - The window slides when a correct frame has arrived; sliding occurs one slot at a time $R_n = R_n + 1$
    - Out-of order packet (Region I and III in receiver window packet) is discarded → need to be resend

Send window for Go-Back-N ARQ

- Possible Sequence number (m=4) division – four and Sliding window size = 2^4-1 = 15



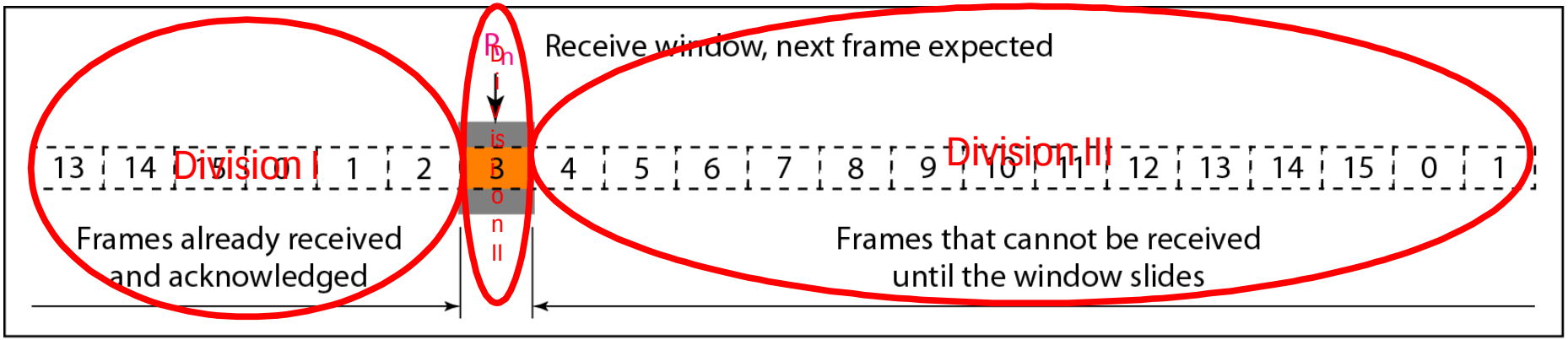
a. Send window before sliding



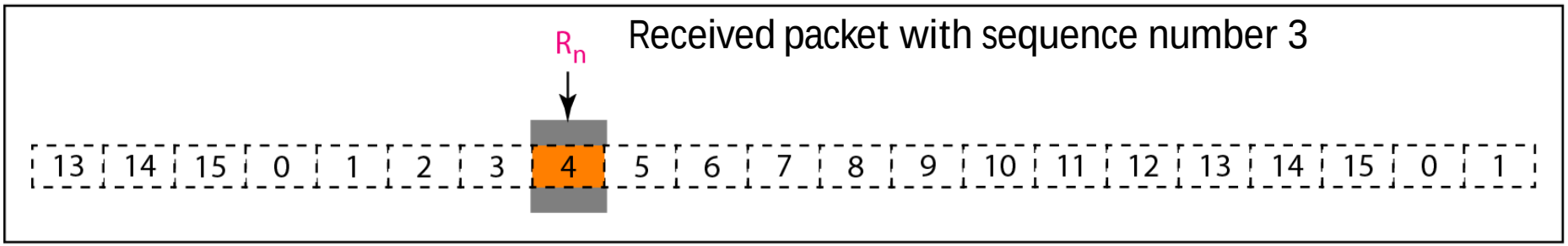
b. Send window after sliding

Receive window for Go-Back-N ARQ

- Possible ACK number division – three and Sliding window size = 1



a. Receive window



b. Window after sliding

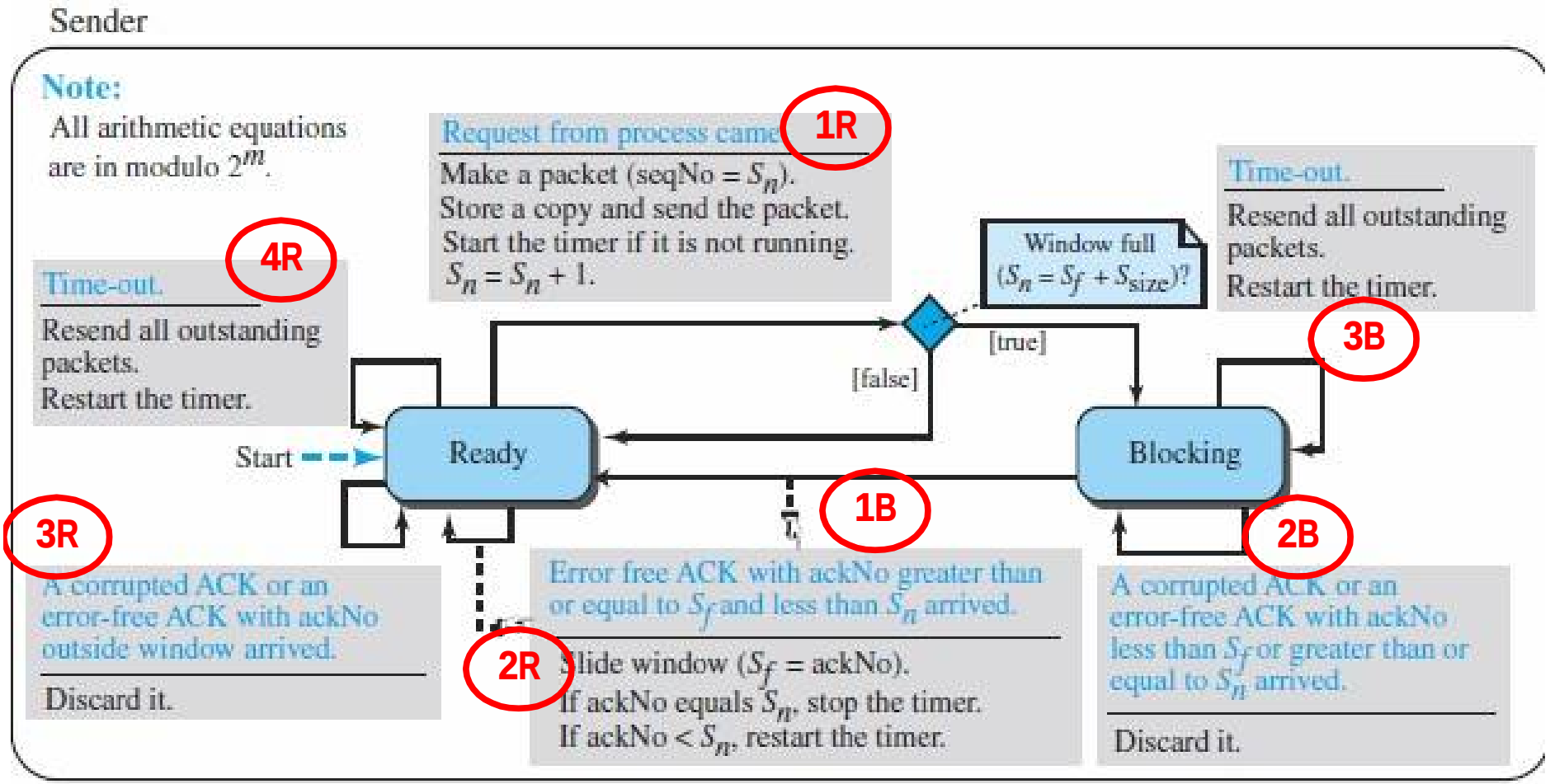
Timer for sent frame

- Multiple frame sent but only one timer (for S_f) used
- As first outstanding packet always expire first
- If S_f timer expires all outstanding packets are represent that is why GBN
- On a time-out the machine goes back N-locations and resents all packets

FSMs for the GBN protocol

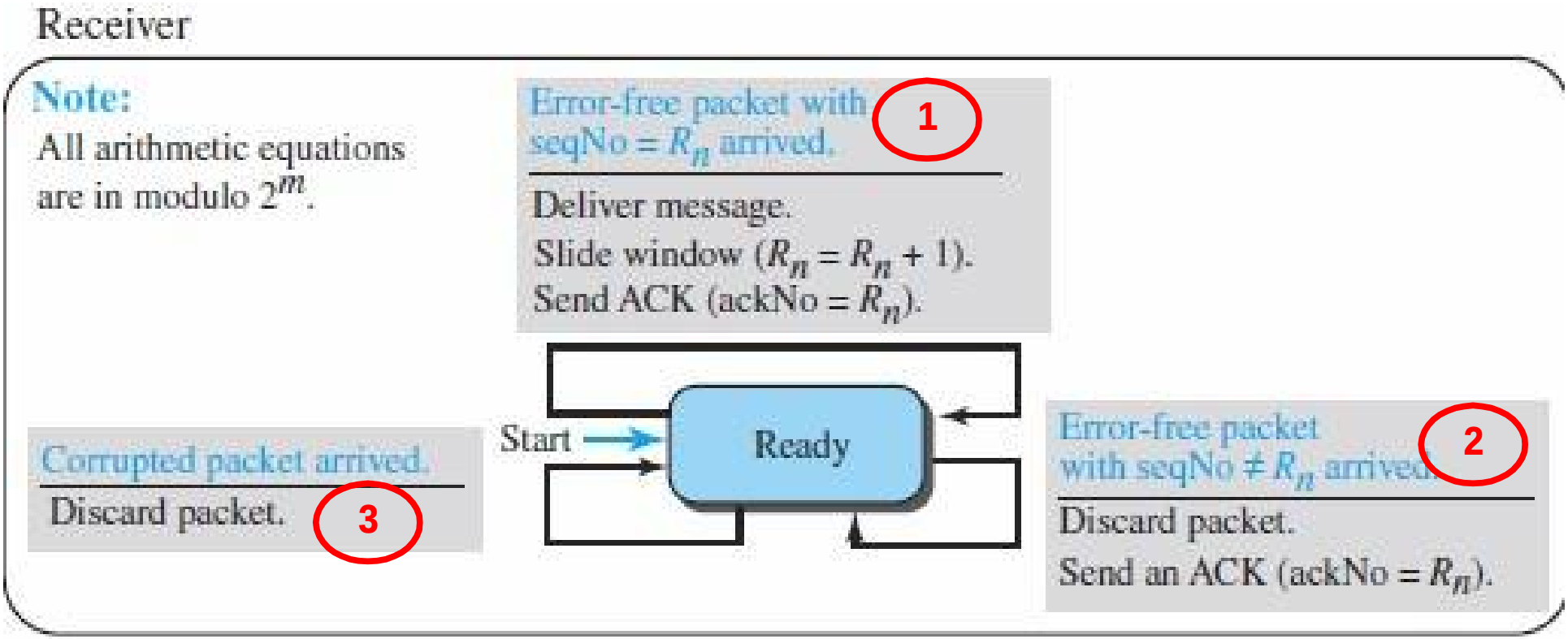
- Modulo 2^m arithmetic
- Sender FSM: two states
  - ready → 4-events can occur → numbered as 1R,2R, 3R, 4R
  - Blocking → 3-events may occur → numbered as 1B,2B, 3B
- The sender starts in the ready state
  - *The two variables are normally initialized to 0 (Sf = Sn = 0)*

Sender FSM


Sender
Note:
All arithmetic equations are in modulo 2^m.
Request from process came
Make a packet (seqNo = S_n).
Store a copy and send the packet.
Start the timer if it is not running.
$S_n = S_n + 1$.
1R
Window full ($S_n = S_f + S_{size}$)?
[true]
[false]
Time-out.
Resend all outstanding packets.
Restart the timer.
3B
4R
Time-out.
Resend all outstanding packets.
Restart the timer.
Start
Ready
Blocking
1B
2B
3R
A corrupted ACK or an error-free ACK with ackNo outside window arrived.
Discard it.
2R
Error free ACK with ackNo greater than or equal to S_f and less than S_n arrived.
Slide window (S_f = ackNo).
If ackNo equals S_n, stop the timer.
If ackNo < S_n, restart the timer.
A corrupted ACK or an error-free ACK with ackNo less than S_f or greater than or equal to S_n arrived.
Discard it.

Receiver FSM

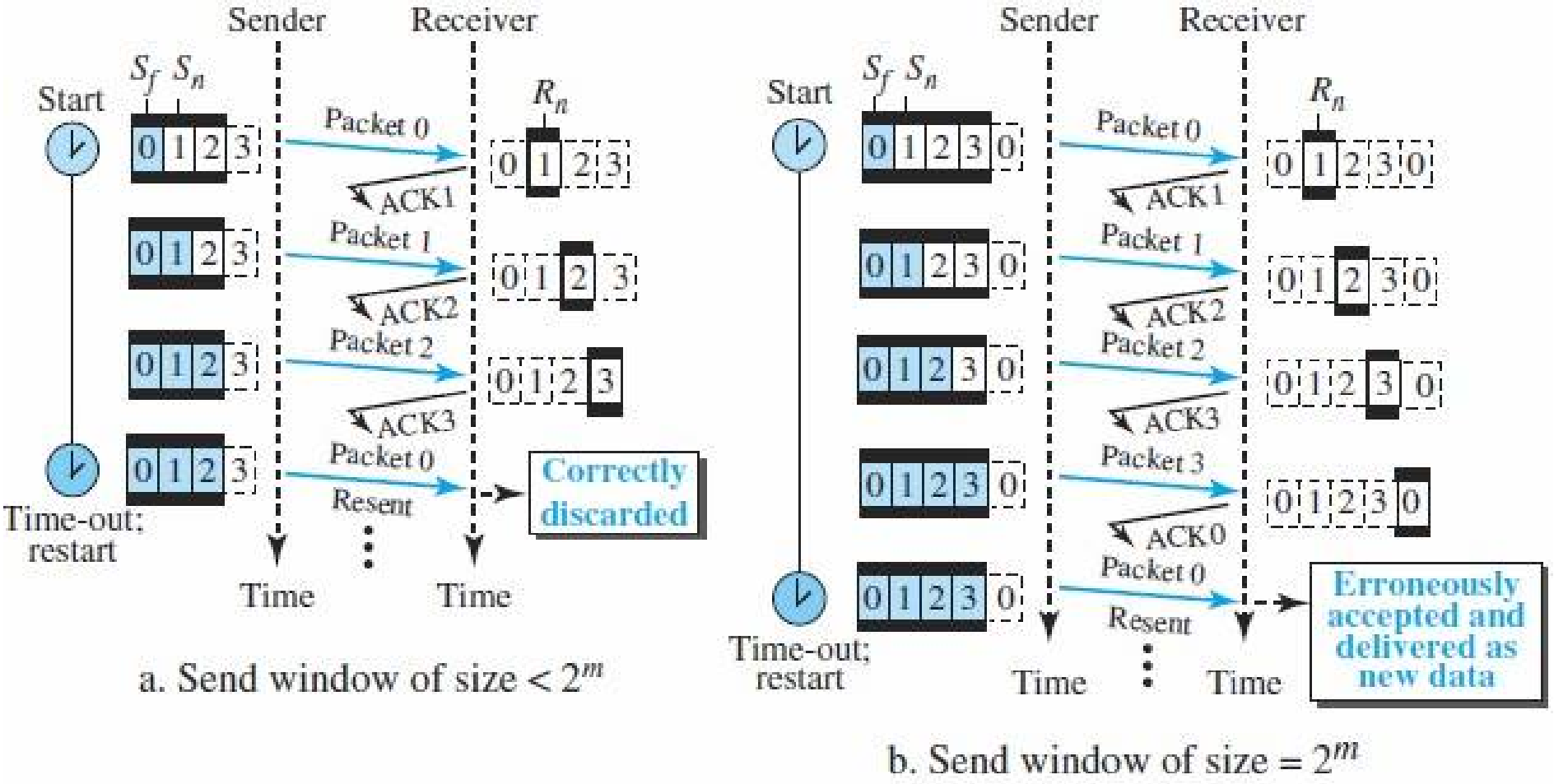
- R_n = 0 initially
- Only one state → 3 states



Size of send window

- Why the size of send window must be less than 2^m?
  - For m=2, window size < 2^m = 3
    - If all 3 - ACK lost, timer expires and all three packets are resend
  - For m=2, window size = 2^m = 4
    - If all 4 - ACK lost, timer expires and all four packets are resend which are accepted as new data erroneously even through duplicate
- **Hence *the size of the send window must be less than* 2^m*;* the size of the receive window is always 1**

Size of send window

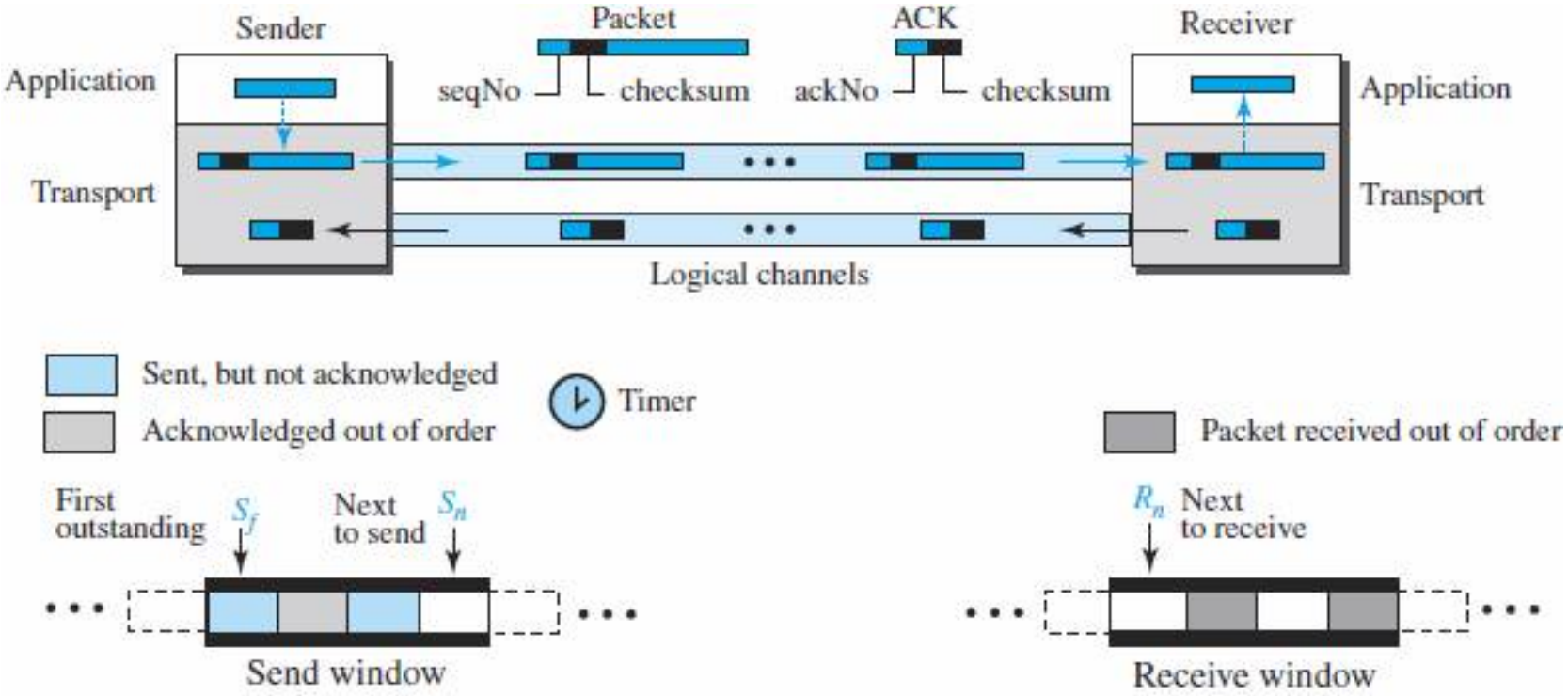


GBN Analysis

- Advantage: Simplifies the process at the receiver
  - Receiver keeps track of only one variable
  - No need to buffer out-of-order packets
    - simply discarded
- Disadvantage: Inefficient if the underlying network protocol loses a lot of packets
  - Each time a single packet is lost or corrupted, the sender resends all outstanding packets, even though some of these packets may have been received safe and sound but out of order
  - If the network layer is losing many packets because of congestion in the network, the resending of all of these outstanding packets makes the congestion worse, and eventually more packets are lost
  - This has an avalanche effect that may result in the total collapse of the network
    - Solution Selective Repeat

Selective-Repeat (SR) protocol

- Name implies, resends only selective packets, those that are actually lost

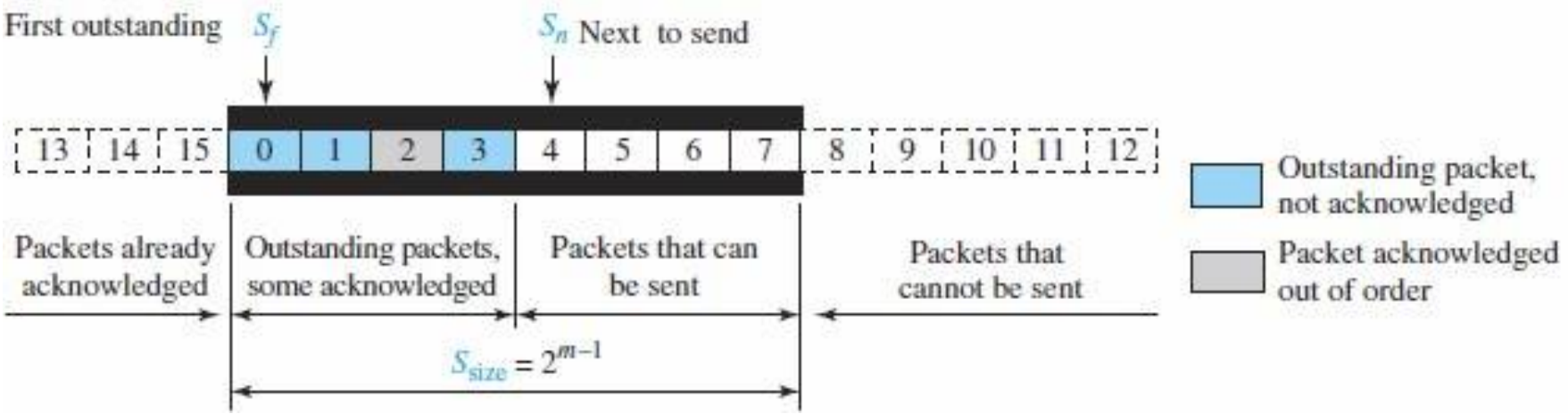


SR protocol windows

- two windows: a send window and a receive window

SR protocol windows	GBN protocol windows
Maximum size of the send window is much smaller (maximum 2^m-1)	*The send window is* 2^m-1
The receive window is the same size as the send window (maximum 2^m-1)	*The receive window is 1*
if m = 4, the sequence numbers go from 0 to 15, but the maximum size of the window is just 8	*if m = 4, the sequence* numbers go from 0 to 15 the size of the window is 15

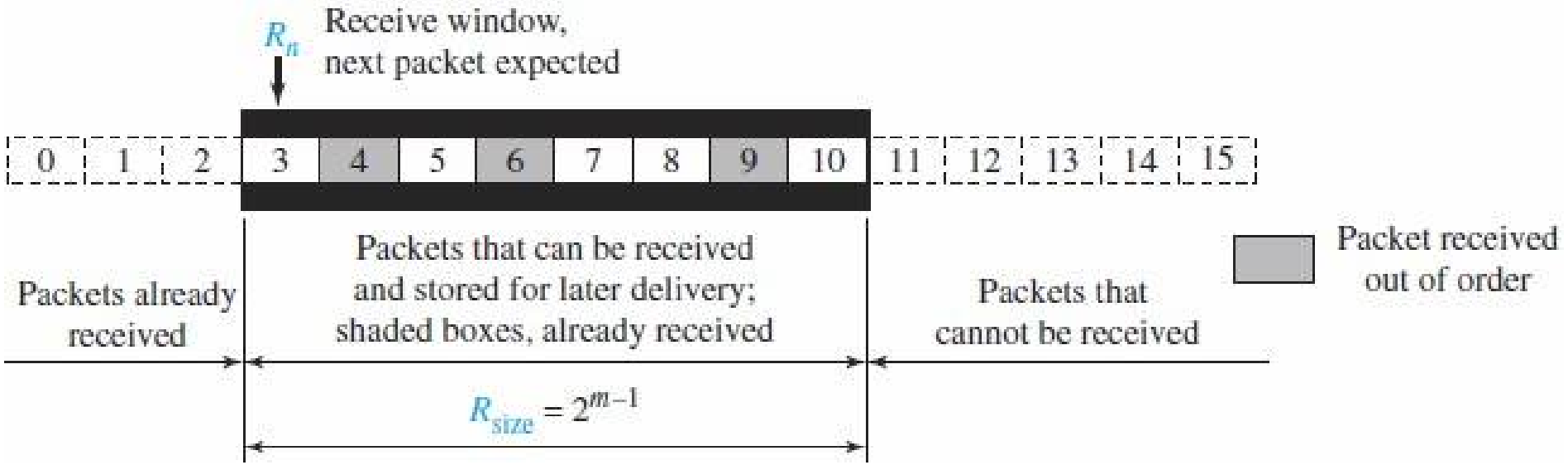
Send window for SR protocol



- Allows as many packets as the size of the receive window to arrive out of order and be kept until there is a set of consecutive packets to be delivered to the application layer
- Because the sizes of the send window and receive window are the same, all the packets in the send packet can arrive out of order and be stored until they can be delivered
- A reliable protocol never delivers packets out of order to the application layer

Receive window for SR protocol

- Shaded slots inside the window define packets that have arrived out of order and are waiting for the earlier transmitted packet to arrive before delivery to the application layer



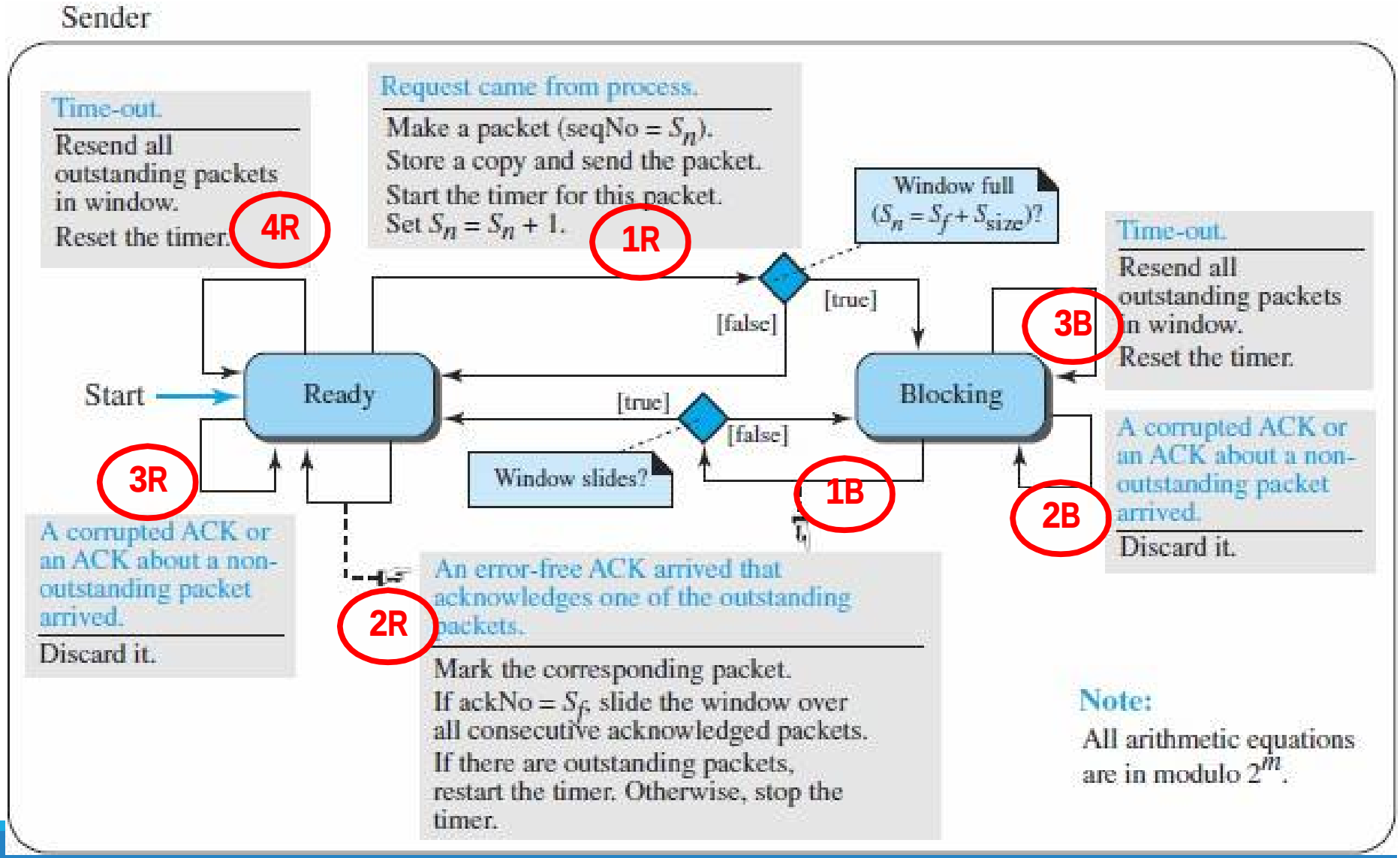
SR protocol *Timer and Acknowledgements*

- **Timers:**
- Theoretically, one timer for each outstanding packet
- When a timer expires, only the corresponding packet is resent
- GBN treats outstanding packets as a group; SR treats them individually
- However, most transport-layer protocols that implement SR use only a single timer

Acknowledgements:

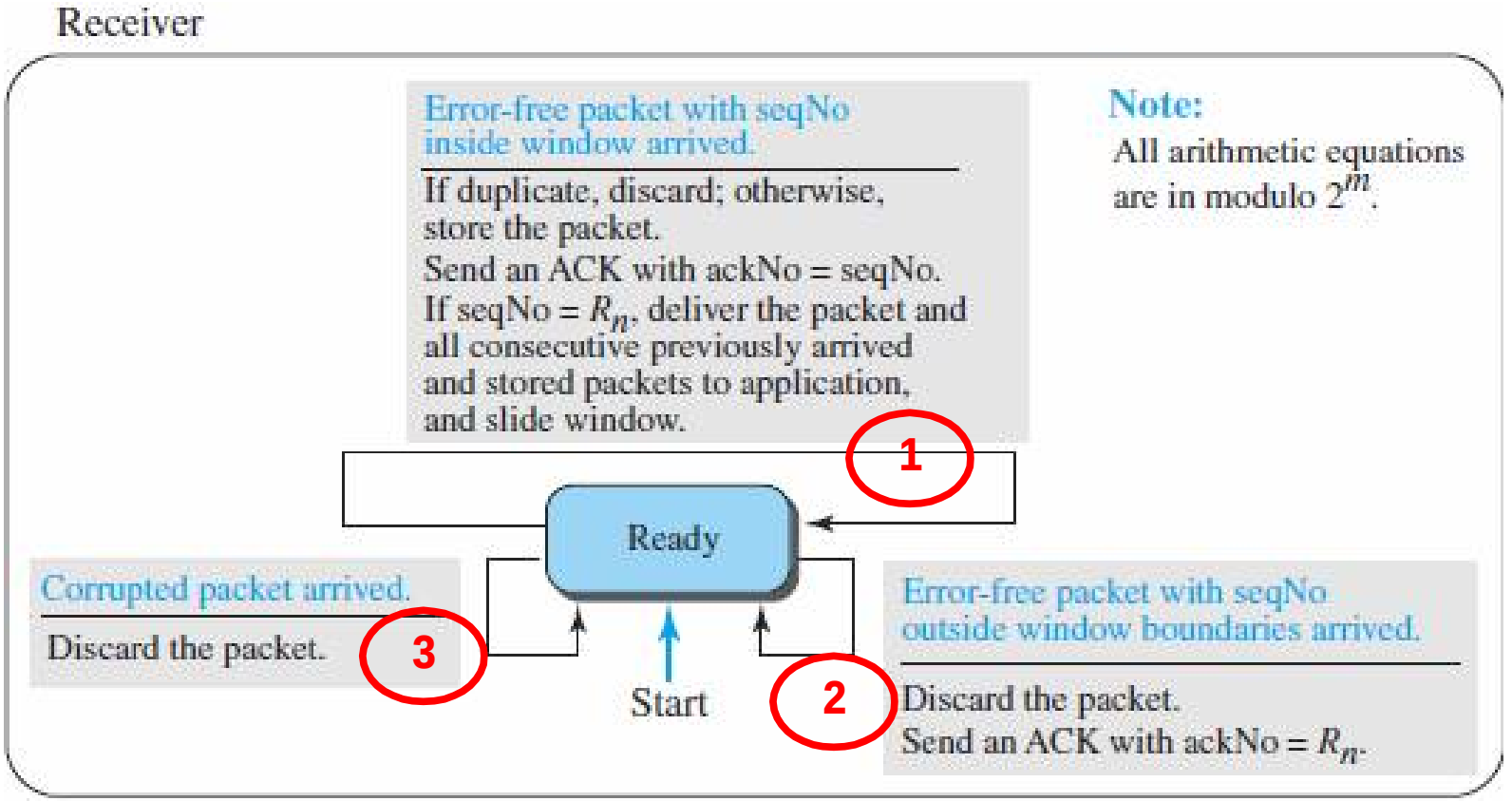
- GBN an ackNo is cumulative (defines the sequence number of the next packet expected, confirming that all previous packets have been received safe and sound)
- The semantics of acknowledgment is different in SR
  - An ackNo defines the sequence number of a single packet that is received safe and sound; there is no feedback for any other

SR Protocol Sender FSM



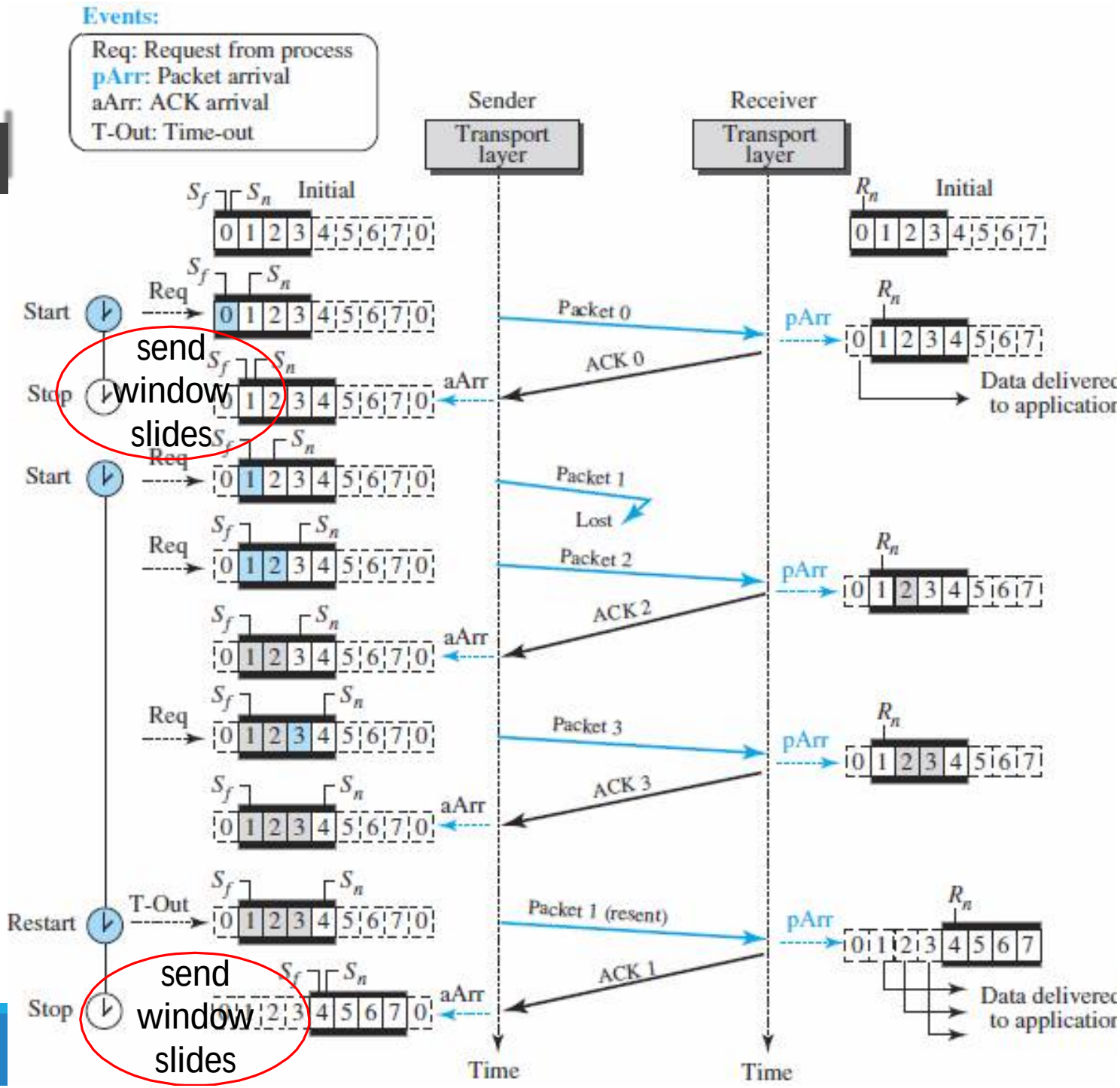
SR Protocol Receiver FSM

- always in the *ready state*



SR Protocol

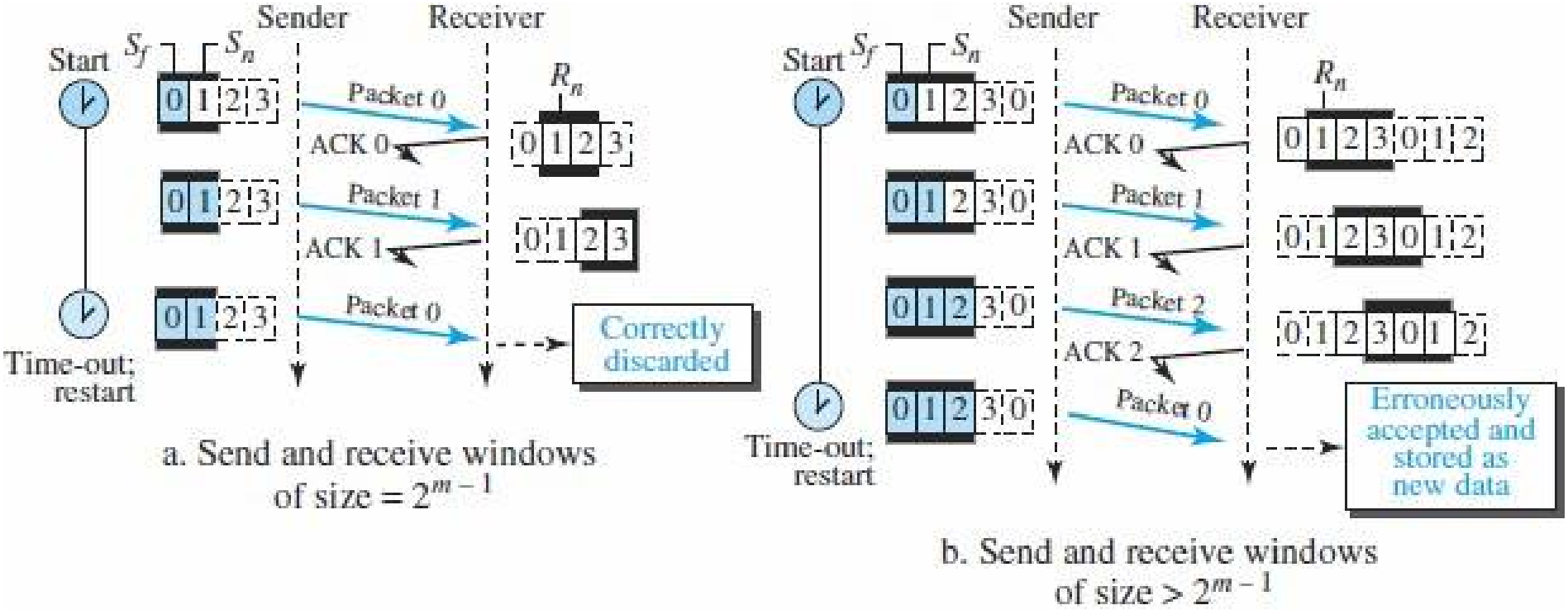
- Packets 0, 1, 2, and 3 are sent. However, packet 1 is lost.



SR Protocol *Window Sizes*

- Why the size of the sender and receiver windows can be at most one-half of 2^m?
- *For m = 2, size of the window is $2^m/2$ or $2^{(m-1)} = 2$*
- If the size of the window is 2 and all acknowledgments are lost, the timer for packet 0 expires and packet 0 is resent. However, the window of the receiver is now expecting packet 2, not packet 0, so this duplicate packet is correctly discarded (the sequence number 0 is not in the window)
- When the size of the window is 3 and all acknowledgments are lost, the sender sends a duplicate of packet 0
  - However, this time, the window of the receiver expects to receive packet 0 (0 is part of the window), so it accepts packet 0, not as a duplicate, but as a packet in the next cycle
  - This is clearly an error

SR Protocol *Window Sizes*



- The size of the sender and receiver window must be at most one-half of 2^m

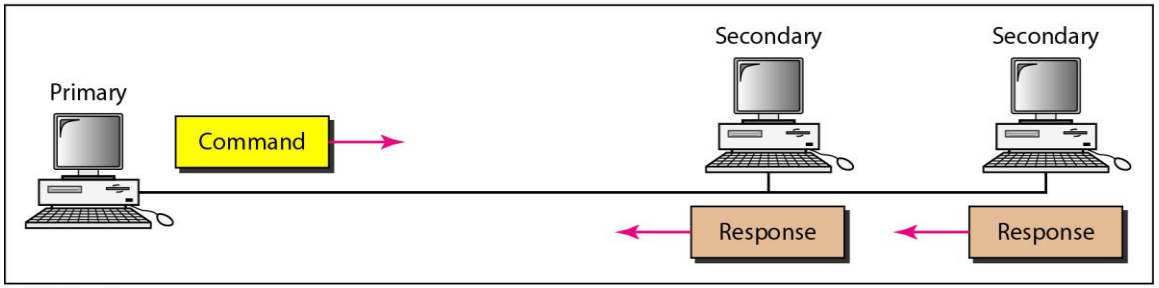
High-level Data Link Control

- High-level Data Link Control (HDLC) is a bit-oriented protocol for communication over point-to-point and multipoint links
  - implements the Stop-and-Wait protocol ARQ mechanisms
- theoretical issue than practical, but is the basis for other practical protocols such as PPP
- Two common transfer modes used in different configurations:
  - *Normal response mode (NRM)*
  - *Asynchronous balanced mode (ABM)*

HDLC: Normal response mode

- *Normal response mode (NRM):* unbalanced station configuration
  - one primary station (send commands) and multiple secondary stations (only respond)
  - used for both point-to-point and multipoint links

a. Point-to-point



b. Multipoint

Asynchronous balanced mode

- Configuration is balanced
- The link is point-to-point, and each station can function as a primary and a secondary (acting as peers)
- This is the common mode today

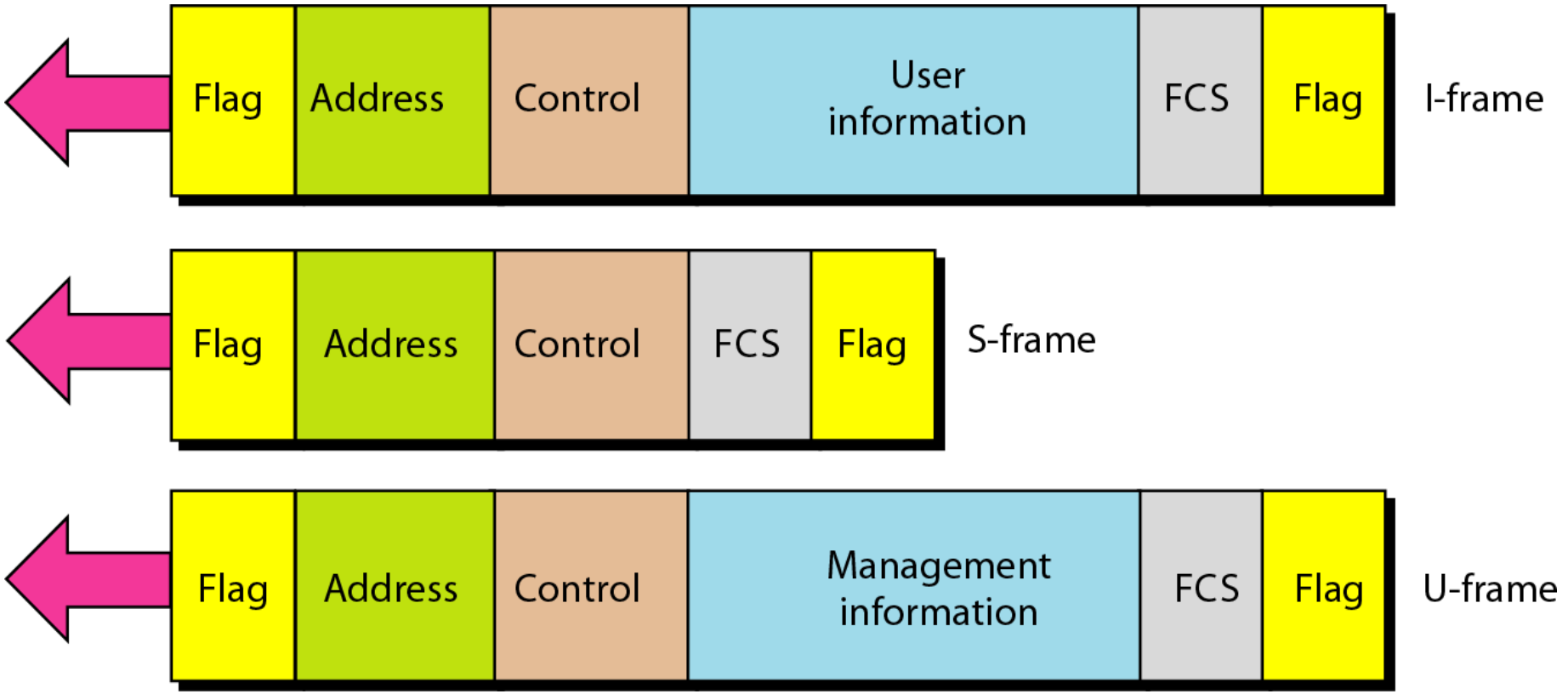
HDLC: Framing

- To provide the flexibility necessary to support all the options possible in the modes and configurations, HDLC defines three types of frames:
  - *Information frames (I-frames)* :
    - designed to carry user data from the network layer
    - include flow- and error-control information (piggybacking)
  - *Supervisory frames (S-frames)* : used only to transport control information
    - used for flow and error control whenever piggybacking is either impossible or inappropriate
  - *Unnumbered frames (U-frames)*: reserved for system management (intended for managing the link itself)
    - used to exchange session management and control information between connected devices
- *Each type* of frame serves as an envelope for the transmission of a different type of message

HDLC frames

- Each frame in HDLC may contain up to six fields
  1. A beginning flag field
  2. An address field
  3. A control field
  4. An information field
  5. A frame check sequence (FCS) field
  6. An ending flag field
- In multiple-frame transmissions, the ending flag of one frame can serve as the beginning flag of the next frame

HDLC frames ...


Flag
Address
Control
User information
FCS
Flag
I-frame
Flag
Address
Control
FCS
Flag
S-frame
Flag
Address
Control
Management information
FCS
Flag
U-frame

HDLC frames ...

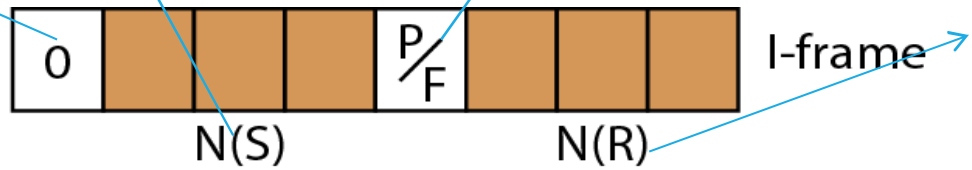
- Flag field: contains synchronization pattern 01111110, which identifies both the beginning and the end of a frame
- Address field: contains the address of the secondary station
  - If a primary station created the frame, it contains a ***to*** address
  - If a secondary station creates the frame, it contains a ***from*** address
  - The address field can be one byte or several bytes long, depending on the needs of the network
- Control field: one or two bytes used for flow and error control
  - Also determines the type of frame and defines its functionality
- Information field: contains the user's data from the network layer or management information
  - Its length can vary from one network to another.
- Frame check sequence (FCS) field : HDLC error detection field
  - can contain either a 2- or 4-byte CRC

Control field format

type : 0 means the frame is an I-frame

*sequence number of the frame (*between 0 and 7 as 3-bit field)

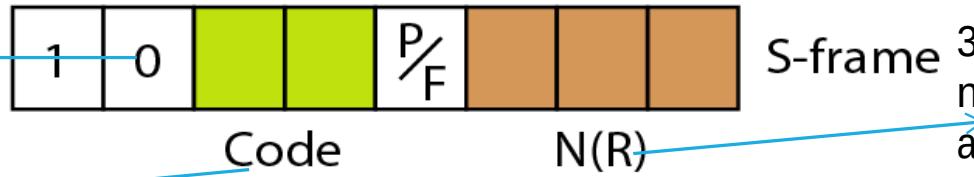
1-bit poll or final: *has* meaning only when bit = 1
Poll : frame is sent by a primary station to a secondary (address field = address of the receiver)
Final : frame is sent by a secondary to a primary (address field = address of the sender)



I-frame

3-bit acknowledgment number when piggybacking

control field means the frame is an S-frame



S-frame

3-bit acknowledgment number or negative acknowledgment number, depending on the type of S-frame

Define the type of S-frame itself

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

Point-to-Point Protocol

- Connect home computers to the server of an Internet using Point-to-Point Protocol (PPP)
- Modem connected to the Internet through a telephone line, which provides the services of the physical layer
- But to control and manage the transfer of data, there is a need for a point-to-point protocol at the data-link layer
- PPP is a byte-oriented protocol

Services Provided by PPP

- Format of the frame to be exchanged between devices
- How two devices can negotiate the establishment of the link and the exchange of data
- Accept payloads from several network layers (not only IP)
- Optional authentication
- The new version of PPP, called *Multilink PPP, provides connections over multiple links*
- Network address configuration
- useful when a home user needs a temporary network address to connect to the Internet

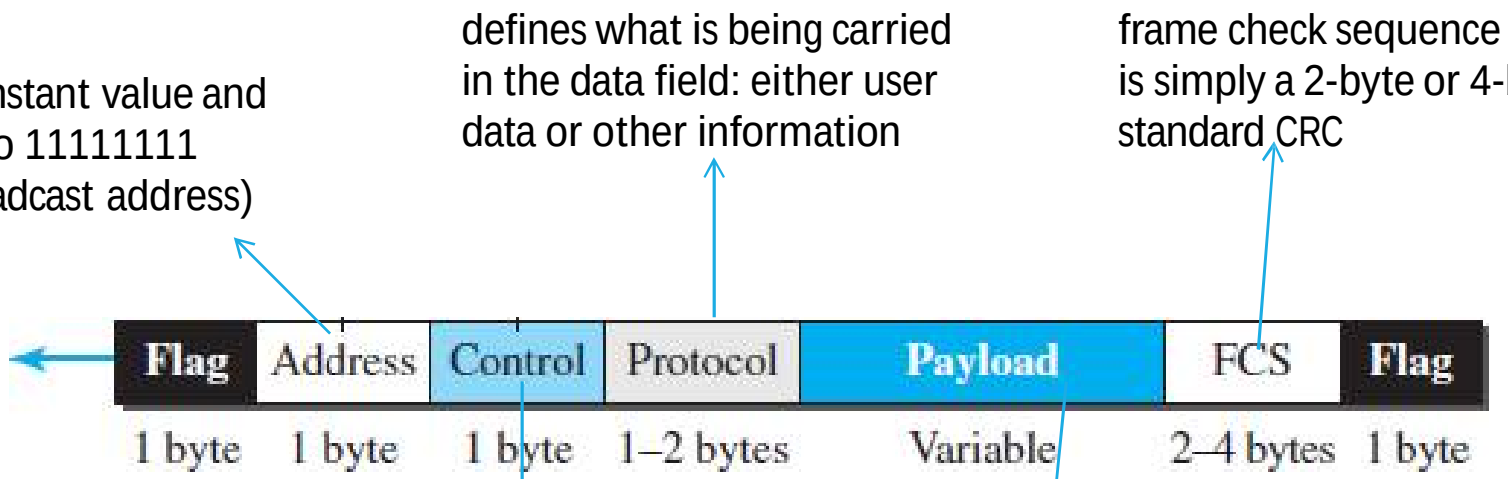
PPP Services

Services Not Provided by PPP

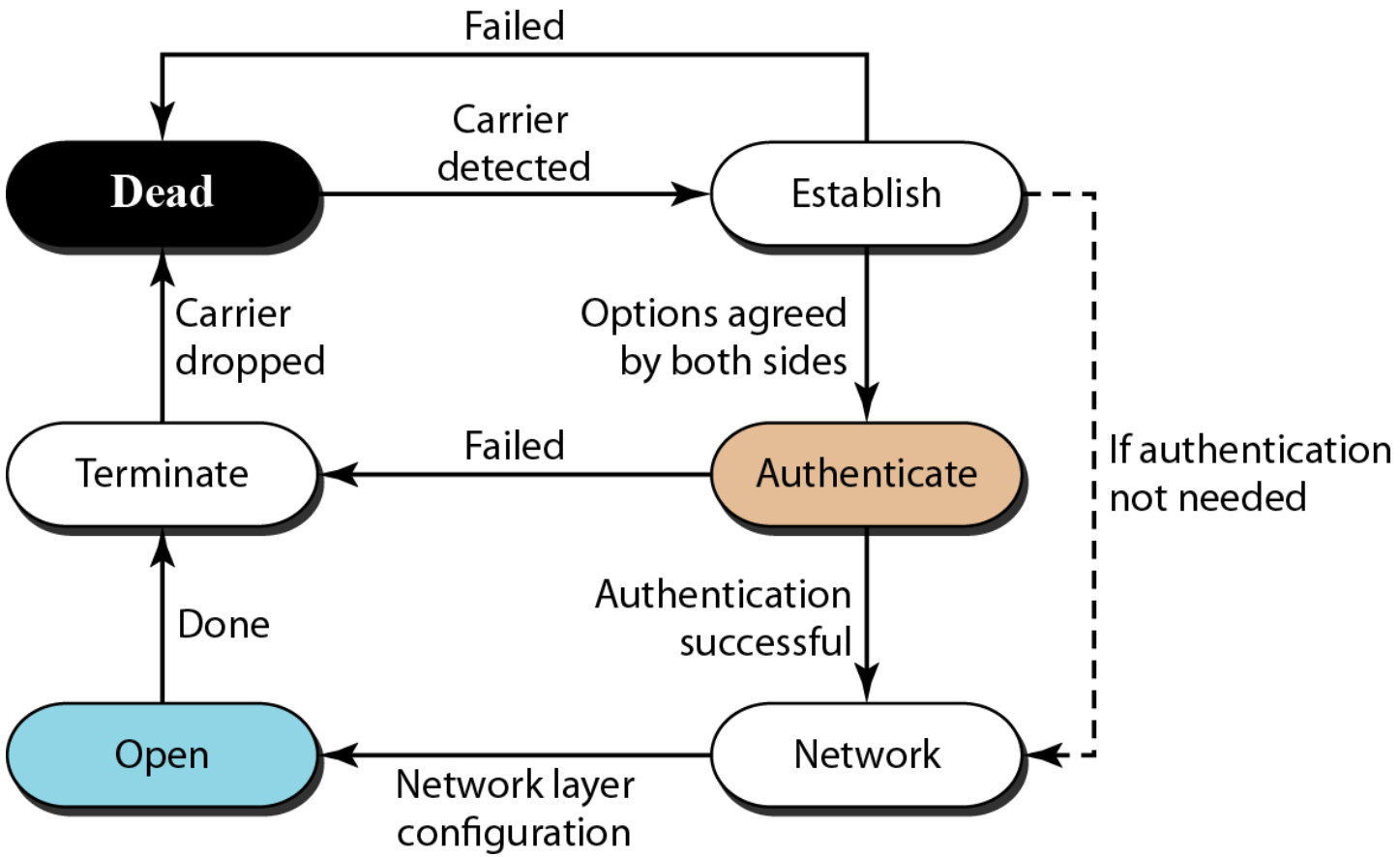
- No flow control - A sender can send several frames one after another with no concern about overwhelming the receiver
- A very simple mechanism for error control - A CRC field is used to detect errors
  - If the frame is corrupted, it is silently discarded; the upper-layer protocol needs to take care of the problem
    - Lack of error control and sequence numbering may cause a packet to be received out of order
- No a sophisticated addressing mechanism to handle frames in a multipoint configuration

PPP frame format

- a character-oriented (or byte-oriented) frame
- starts and ends with a 1-byte flag with the bit pattern 01111110



Transition Phases (FSM)

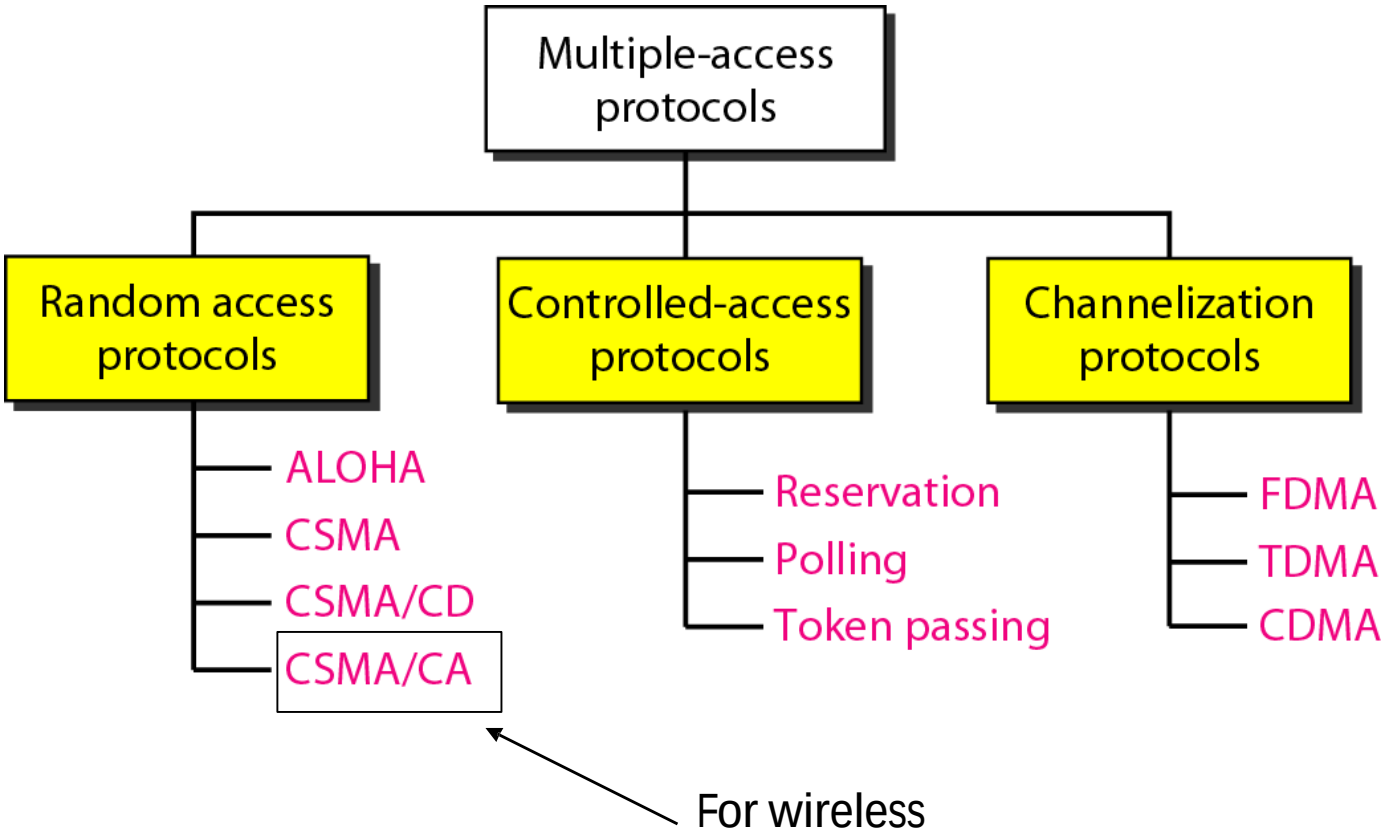


END

Media Access Control (MAC)

CS44 Data Communications

Taxonomy of multiple-access protocols

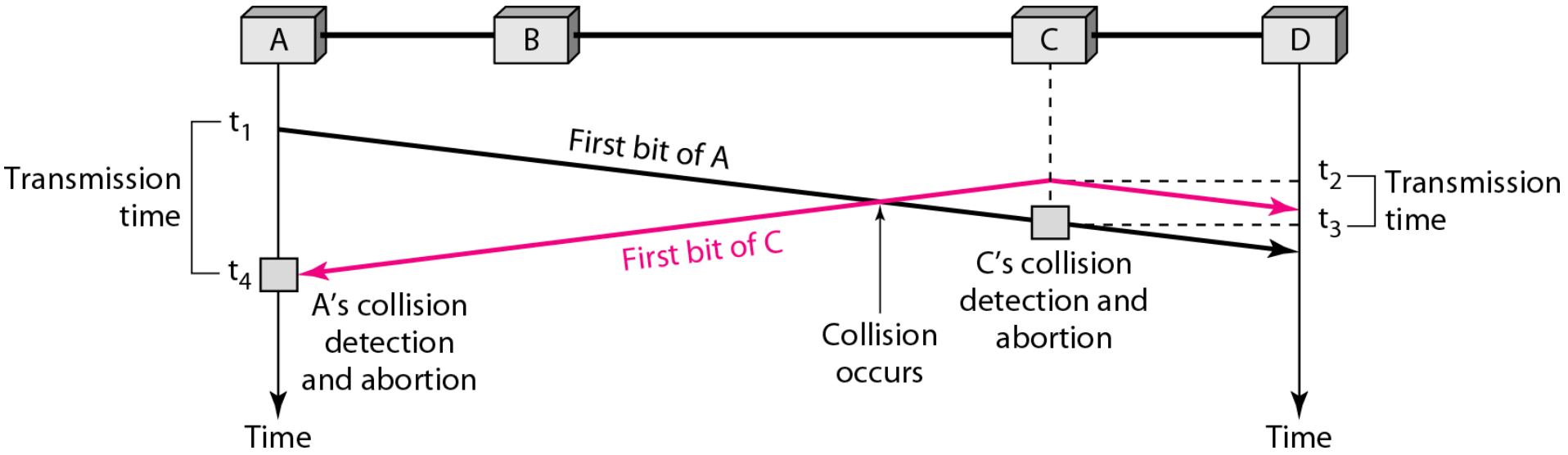


Carrier sense multiple access with collision detection (CSMA/CD)

- The CSMA method does not specify the procedure following a collision
- CSMA/CD augments the algorithm to handle the collision
  - A station monitors the medium after it sends a frame to see if the transmission was successful
  - If so, the station is finished
  - If there is a collision, the frame is sent again

CSMA/CD ...

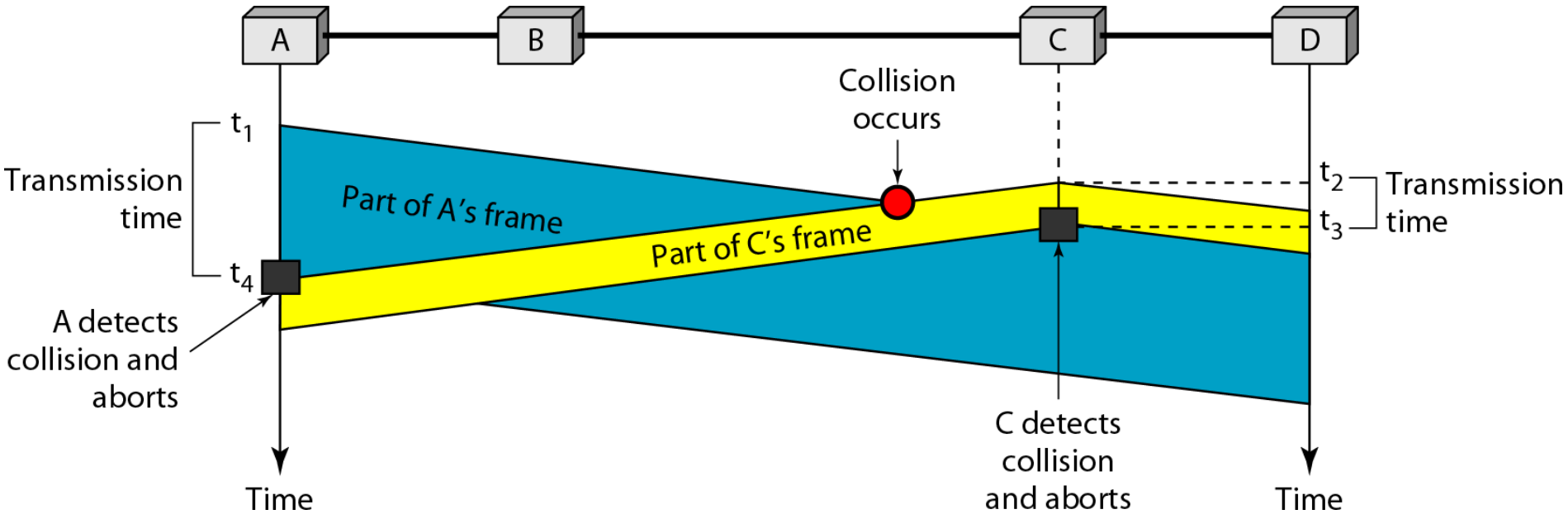
- Look at the first bits transmitted by the two stations involved in the collision
  - each station continues to send bits in the frame until it detects the collision



CSMA/CD ...

- At time t1, station A has executed its persistence procedure and starts sending the bits of its frame
- At time t2, station C has not yet sensed the first bit sent by A
- Station C executes its persistence procedure and starts sending the bits in its frame, which propagate both to the left and to the right
- The collision occurs sometime after time t2
- Station C detects a collision at time t3 when it receives the first bit of A's frame
- Station C immediately (or after a short time, but we assume immediately) aborts transmission
- Station A detects collision at time t4 when it receives the first bit of C's frame - immediately aborts transmission
- A transmits for the duration t4 – t1; C transmits for the duration t3 – t2

CSMA/CD - Collision and abortion



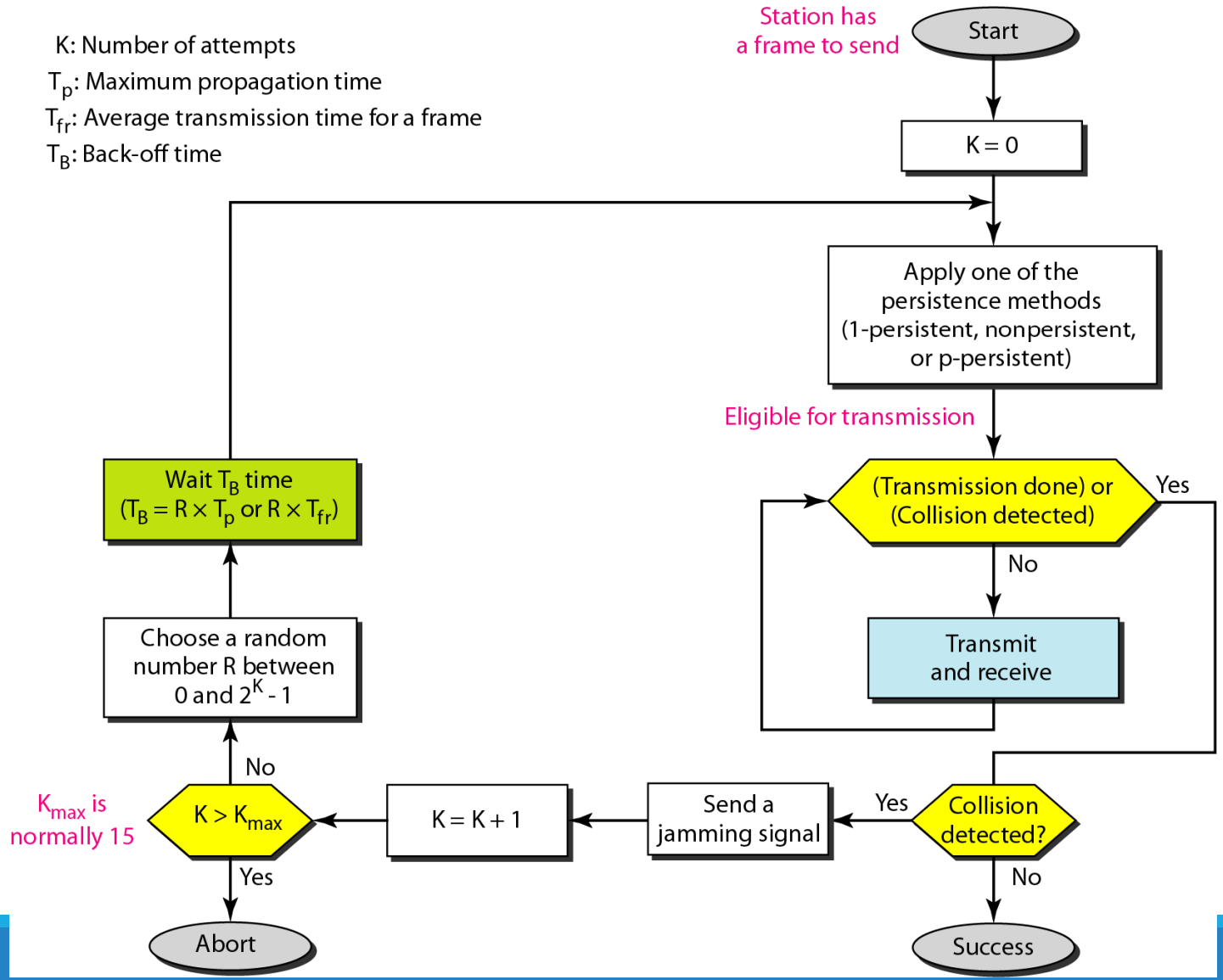
CSMA/CD -*Minimum Frame Size*

- Need a restriction on the frame size
- Before sending the last bit of the frame, the sending station must detect a collision, if any, and abort the transmission
  - because the station, once the entire frame is sent, does not keep a copy of the frame and does not monitor the line for collision detection
  - The frame transmission time T_{fr} *must be at least two times the maximum propagation time* T_p
  - Worst-case scenario. If the two stations involved in a collision are the maximum distance apart, the signal from the first takes time T_p *to reach* the second, and the effect of the collision takes another time T_P *to reach the first*
  - *So the* requirement is that the first station must still be transmitting after $2T_p$

CSMA/CD Example

- A network using CSMA/CD has a bandwidth of 10 Mbps
- If the maximum propagation time (including the delays in the devices and ignoring the time needed to send a jamming signal is 25.6 μs, what is the minimum size of the frame?
- **Solution**
  - The minimum frame transmission time is $T_{fr} = 2 \times Tp = 51.2\ \mu s$
  - *The worst case, a* station needs to transmit for a period of 51.2 μs to detect the collision
  - The minimum size of the frame is 10 Mbps × 51.2 μs = 512 bits or 64 bytes
    - This is actually the minimum size of the frame for Standard Ethernet

Flow diagram for CSMA/CD

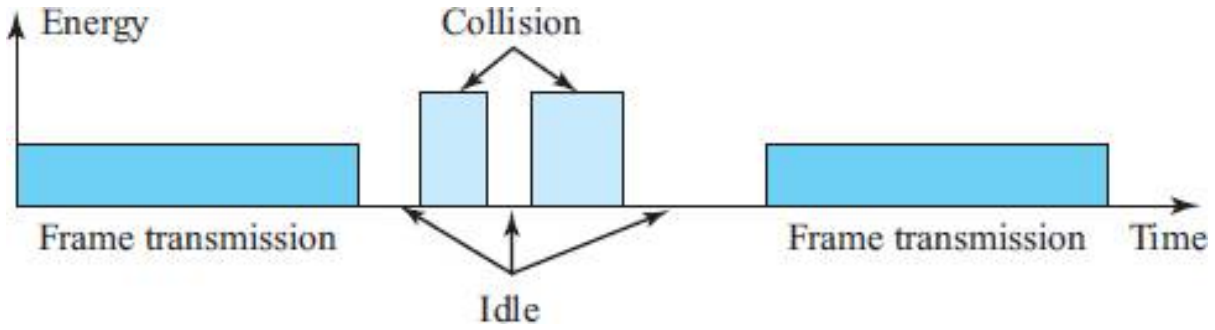


Flow diagram for CSMA/CD

- Similar to the one for the ALOHA protocol with 3-differences
  - addition of the persistence process - sense the channel before start sending the frame by using one of the persistence processes (nonpersistent, 1-persistent, or *p-persistent)*
  - frame transmission
    - In ALOHA, first transmit the entire frame and then wait for an acknowledgment
    - In CSMA/CD, transmission and collision detection are continuous processes (shown as a loop)
      - constantly monitor in order to detect one of two conditions to stop transmission
        - either transmission is finished or
        - a collision is detected
      - On loop exit, if a collision has not been detected, it means that transmission is complete; the entire frame is transmitted. Otherwise, a collision has occurred
  - Sending of a short **jamming signal to make sure that all** other stations become aware of the collision

CSMA/CD - *Energy Level*

- The level of energy in a channel can have three values:
  - Zero - the channel is idle
  - Normal - station has successfully captured the channel and is sending its frame
  - Abnormal –there is a collision and the level of the energy is twice the normal level
- A station that has a frame to send or is sending a frame needs to monitor the energy level to determine if the channel is idle, busy, or in collision mode

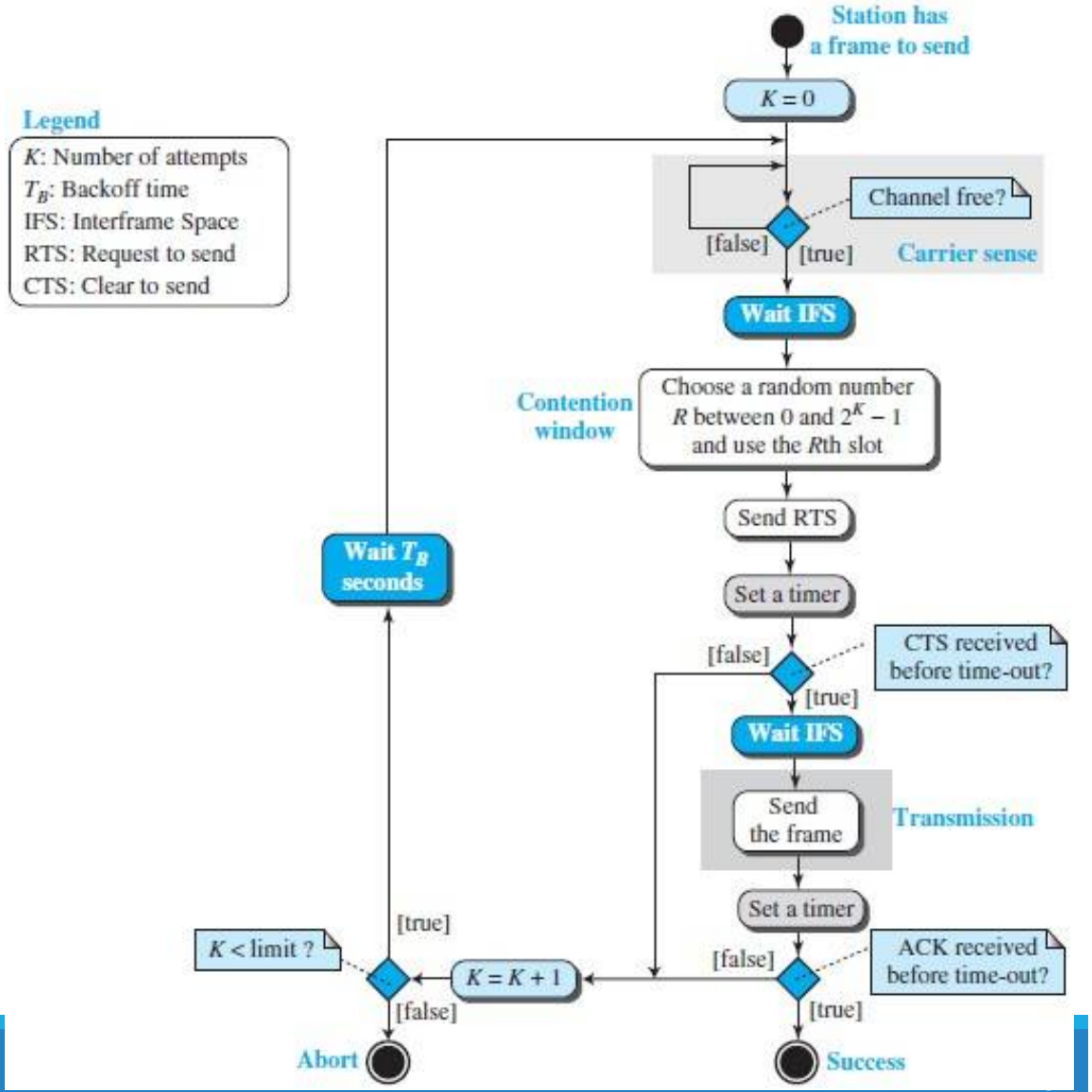


CSMA/CD - *Throughput*

- greater than that of pure or slotted ALOHA
- The maximum throughput occurs at a different value of *G and is based on the persistence* method and the value of *p in the p-persistent approach*
  - *For the 1-persistent method, the* maximum throughput is around 50 percent when *G = 1*
  - *For the nonpersistent method,* the maximum throughput can go up to 90 percent when *G is between 3 and 8*
- *Traditional Ethernet is a* broadcast LAN that used the 1-persistence method to control access to the common media with the data rate of 10 Mbps

Carrier sense multiple access with collision avoidance - CSMA/CA

- **Invented** for wireless networks
- Collisions are avoided through the use of CSMA/CA's three strategies:
  - the interframe space
  - the contention window
  - acknowledgments



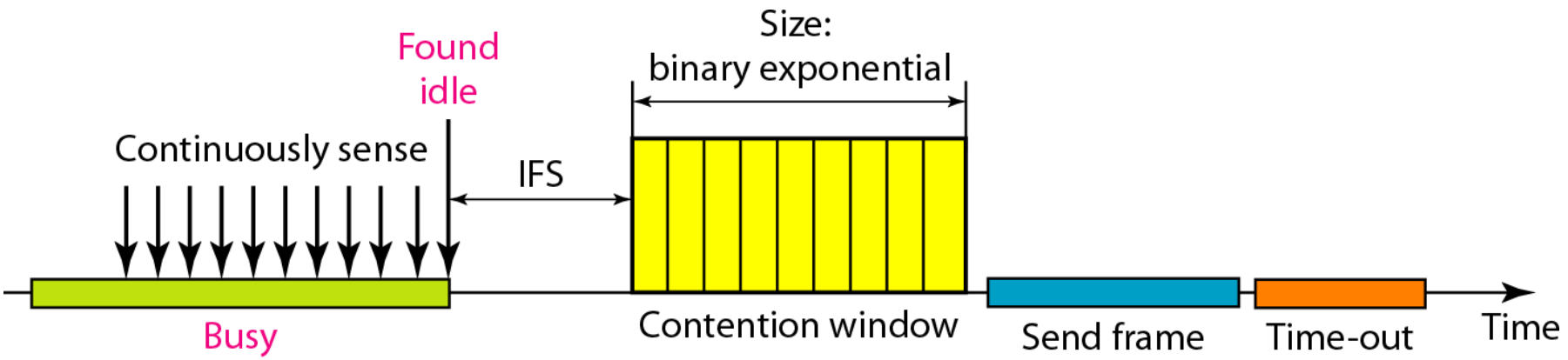
CSMA/CA- *Interframe Space (IFS)*

- *Avoids collisions by deferring transmission even* if the channel is found idle
- When an idle channel is found, the station waits for a period of *IFS* time
- IFS time allows the front of the transmitted signal by the distant station to reach this station
- After waiting an IFS time, if the channel is still idle, the station can send, but it still needs to wait a time equal to the contention window
- The IFS variable can also be used to prioritize stations or frame types
  - For example, a station that is assigned a shorter IFS has a higher priority

CSMA/CA - *Contention Window*

- ***An amount of time divided into*** slots
- A station that is ready to send chooses a random number of slots as its wait time
- The number of slots in the window changes according to the binary exponential backoff strategy
  - It is set to one slot the first time and then doubles each time the station cannot detect an idle channel after the IFS time
  - Very similar to the *p-persistent method except that a random outcome defines the* number of slots taken by the waiting station
- The station needs to sense the channel after each time slot
- However, if the station finds the channel busy, it does not restart the process; it just stops the timer and restarts it when the channel is sensed as idle
- This gives priority to the station with the longest waiting time

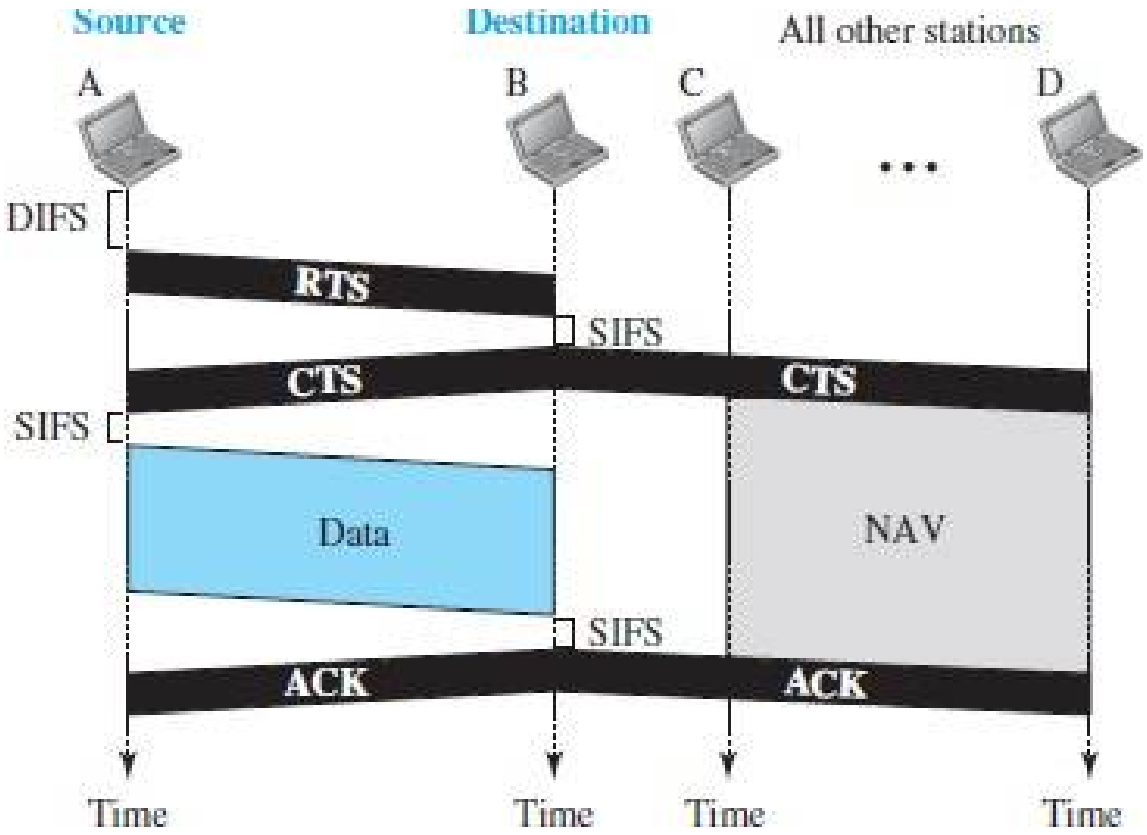
CSMA/CA - Timing



CSMA/CA - *Acknowledgment*

- With all these precautions, there still may be a collision resulting in destroyed data
- In addition, the data may be corrupted during the transmission
- The positive acknowledgment and the time-out timer can help guarantee that the receiver has received the frame

CSMA/CA -*Frame Exchange Time Line*



CSMA/CA -*Frame Exchange Time Line*

- Exchange of data and control frames in time.
  1. Before sending a frame, the source station senses the medium by checking the energy level at the carrier frequency
     a. The channel uses a persistence strategy with backoff until the channel is idle
     b.After the station is found to be idle, the station waits for a period of time called the *DCF interframe space (DIFS); then the station sends a control frame called* the *request to send (RTS).*
  2. After receiving the RTS and waiting a period of time called the *short interframe space (SIFS), the destination station sends a control frame, called the clear to send (CTS), to the source station. This control frame indicates that the destination* station is ready to receive data.
  3. The source station sends data after waiting an amount of time equal to SIFS.
  4. The destination station, after waiting an amount of time equal to SIFS, sends an acknowledgment to show that the frame has been received. Acknowledgment is needed in this protocol because the station does not have any means to check for the successful arrival of its data at the destination. On the other hand, the lack of collision is a kind of indication to the source that data have arrived.

CSMA/CA - *Network Allocation Vector*

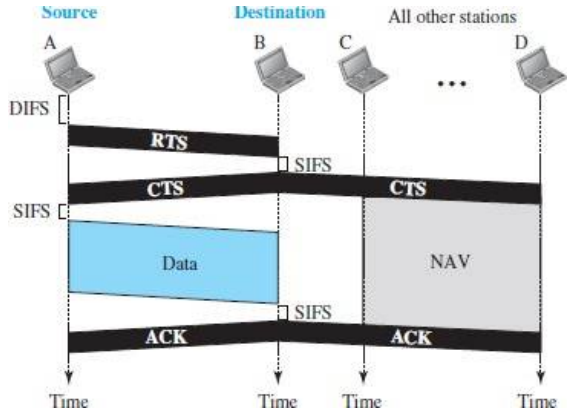
- How do other stations defer sending their data if one station acquires access?
- how is the *collision avoidance aspect of this protocol accomplished?*
- *Use key* feature *Network Allocation Vector (NAV):*
  - When a station sends an RTS frame, it includes the duration of time that it needs to occupy the channel
  - The stations that are affected by this transmission create a timer called a **network allocation vector (NAV) that shows how much time must pass before** these stations are allowed to check the channel for idleness
  - Each time a station accesses the system and sends an RTS frame, other stations start their NAV

CSMA/CA -*Collision During Handshaking*

- *handshaking period* - time when RTS or CTS control frames are in transition
- What happens if there is a collision during *handshaking period ?*
- *Indicates two or more stations may try to* send RTS frames at the same time which may collides
- No mechanism for collision detection, the sender assumes there has been a collision if it has not received a CTS frame from the receiver
- The backoff strategy is employed, and the sender tries again

CSMA/CA - Hidden-Station Problem

- Use of the handshake frames (RTS and CTS)
- The RTS message from B reaches A, but not C.
- However, because both B and C are within the range of A, the CTS message, which contains the duration of data transmission from B to A, reaches C
- Station C knows that some hidden station is using the channel and refrains from transmitting until that duration is over



Controlled access

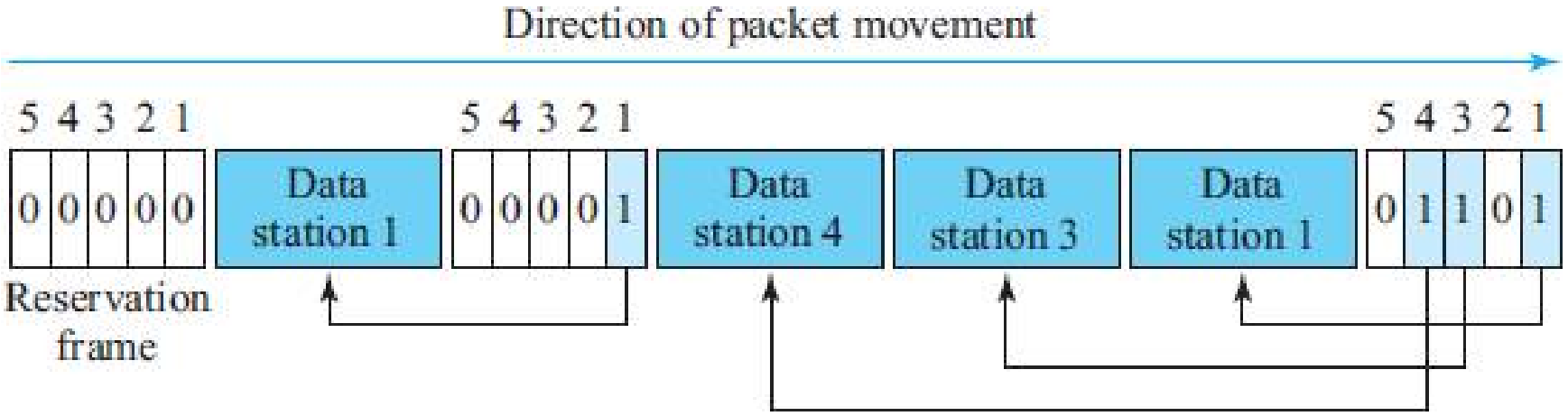
- **The stations consult one another to find which station has the right** to send
- A station cannot send unless it has been authorized by other stations
- Three controlled-access methods
  - **Reservation method**
  - **Polling**
  - **Token Passing**

Reservation method

- **A station needs to make a reservation before sending data**
- Time is divided into intervals
- A reservation frame precedes the data frames sent in each interval.
- If there are *N stations in the system, there are exactly N reservation minislots in the* reservation frame
- Each minislot belongs to a station - When a station needs to send a data frame, it makes a reservation in its own minislot
- The stations that have made reservations can send their data frames after the reservation frame

Reservation access method

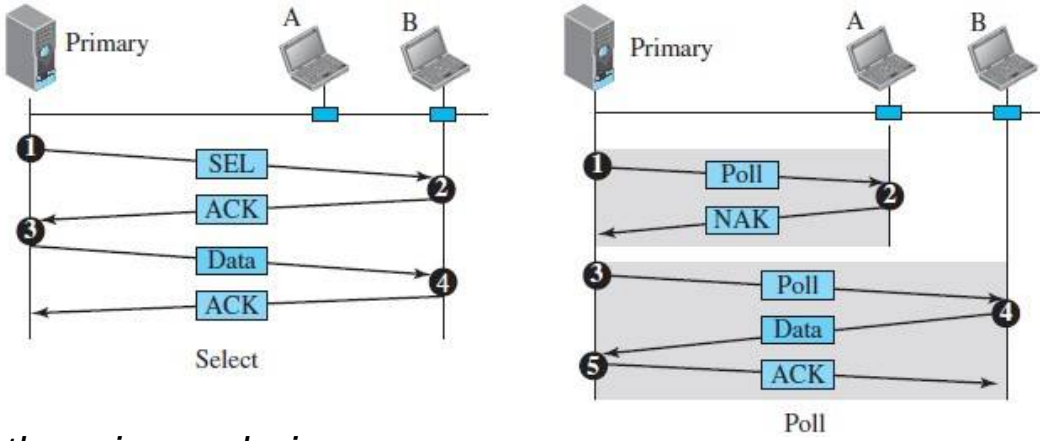
- Five stations and a five-minislot reservation frame
- In the first interval, only stations 1, 3, and 4 have made reservations
- In the second interval, only station 1 has made a reservation



Polling

- Works with topologies in which one device is designated as a primary station that controls the link and the other devices are secondary stations that follow its instructions
- All data exchanges must be made through the primary device even when the ultimate destination is a secondary device
- Primary device - determine which device is allowed to use the channel at a given time and always the initiator of a session
- uses poll and select functions to prevent collisions
- Drawback - if the primary station fails, the system goes down

Polling ...



- *used whenever the primary device has something to send.*
- *Primary* does not know whether the target device is prepared to receive
- Primary alert the secondary about the upcoming transmission and wait for an acknowledgment of the secondary's ready status using select (SEL) frame
- One field of SEL includes the address of the intended secondary

- *Used by the primary device to solicit transmissions from the secondary* devices
- When the primary is ready to receive data, it must ask (poll) each device in turn if it has anything to send
- When the first secondary is approached, it responds either with a AK frame if it has nothing to send or with data if it does
- If the response is negative (a NAK frame), then the primary polls the next secondary in the same manner until it finds one with data to send
- When the response is positive (a data frame), the primary reads the frame and returns an acknowledgment (ACK frame), verifying its receipt

Token Passing

- **The stations in a network are organized in a logical ring where** each station has a *predecessor and a successor*
- *The predecessor* is the station which is logically before the station in the ring; the successor is the station which is after the station in the ring
- The current station is the one that is accessing the channel now
- The right to this access has been passed from the predecessor to the current station then to the successor when the current station has no more data to send

Token Passing

how is the right to access the channel passed from one station to another?

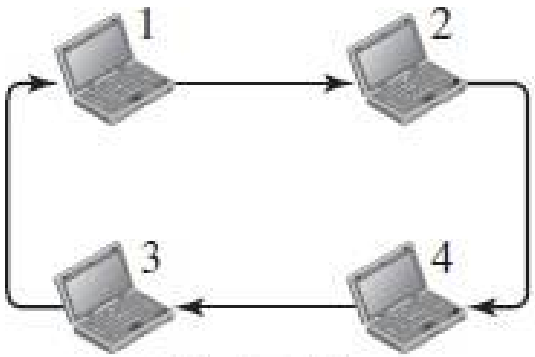
- A special packet called a ***token circulates through the ring***
- ***The possession*** of the token gives the station the right to access the channel and send its data
- When a station has some data to send, it waits until it receives the token from its predecessor
- It then holds the token and sends its data
- When the station has no more data to send, it releases the token, passing it to the next logical station in the ring
- The station cannot send data until it receives the token again in the next round
- When a station receives the token and has no data to send, it just passes the data to the next station

Token management

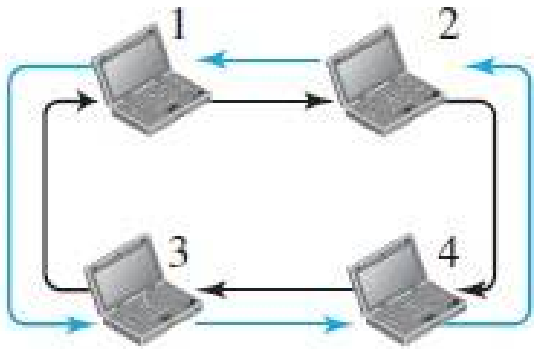
- Stations must be limited
- The token must be monitored to ensure it has not been lost or destroyed
  - For example, if a station that is holding the token fails, the token will disappear from the network
- Assign priorities to the stations and to the types of data being transmitted
- make low-priority stations release the token to high-priority stations

Logical Ring

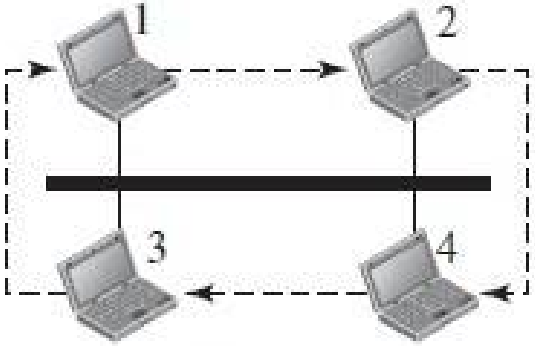
- Four different physical topologies that can create a logical ring



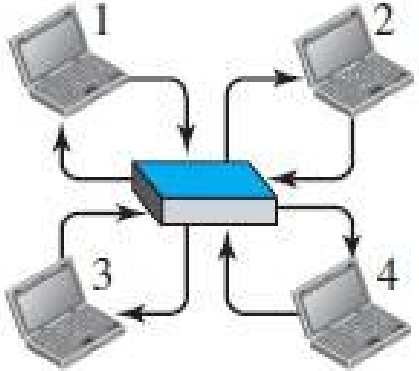
a. Physical ring



b. Dual ring



c. Bus ring



d. Star ring

Physical Ring Topology

- When a station sends the token to its successor, the token cannot be seen by other stations; the successor is the next one in line
- Token does not have to have the address of the next successor
- Problem - if one of the links (the medium between two adjacent stations) fails, the whole system fails

Dual Ring Topology

- Uses a second (auxiliary) ring which operates in the reverse direction compared with the main ring
- The second ring is for emergencies only (such as a spare tire for a car) - If one of the links in the main ring fails, the system automatically combines the two rings to form a temporary ring
- After the failed link is restored, the auxiliary ring becomes idle again
- Each station needs to have two transmitter ports and two receiver ports
- The high-speed Token Ring networks called *FDDI (Fiber Distributed Data Interface) and CDDI (Copper Distributed Data Interface) use this topology*

Bus Ring Topology – Token Bus

- Stations are connected to a single cable called a *bus-make a logical ring, because each station knows* the address of its successor (and also predecessor for token management purposes)
- When a station has finished sending its data, it releases the token and inserts the address of its successor in the token
- Only the station with the address matching the destination address of the token gets the token to access the shared media
- The Token Bus LAN, standardized by IEEE, uses this topology

Star Ring Topology

- Physical topology is a star where a hub acts as the connector
- The wiring inside the hub makes the ring; the stations are connected to this ring through the two wire connections
- This topology makes the network less prone to failure because if a link goes down, it will be bypassed by the hub and the rest of the stations can operate
- Also adding and removing stations from the ring is easier
- Still used in the Token Ring LAN designed by IBM

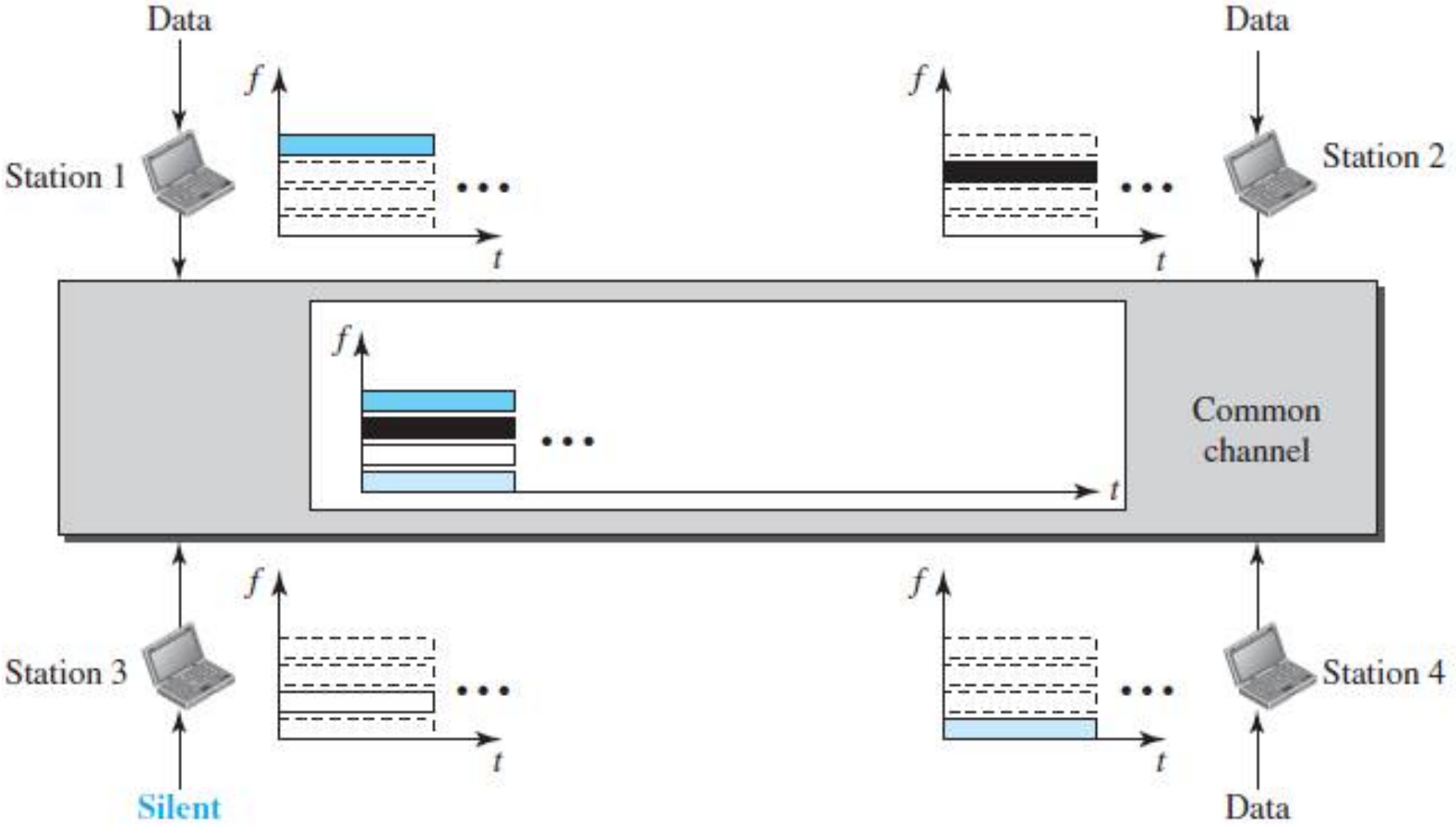
Channelization/ channel partition

- ***A multiple-access*** method in which the available bandwidth of a link is shared in time, frequency, or through code, among different stations
- Three channelization protocols
  - FDMA : Frequency-division multiple access
  - TDMA :Time-division multiple access
  - CDMA : Code-division multiple access

Frequency-division multiple access (FDMA)

- **Available bandwidth is divided** into frequency bands
- Each station is allocated a band to send its data
- Each band is reserved for a specific station, and it belongs to the station all the time
- Each station also uses a bandpass filter to confine the transmitter frequencies
- To prevent station interferences, the allocated bands are separated from one another by small *guard bands*

FDMA ...


Data
Data
Station 1
Station 2
f
t
Common
channel
Station 3
Station 4
Silent
Data

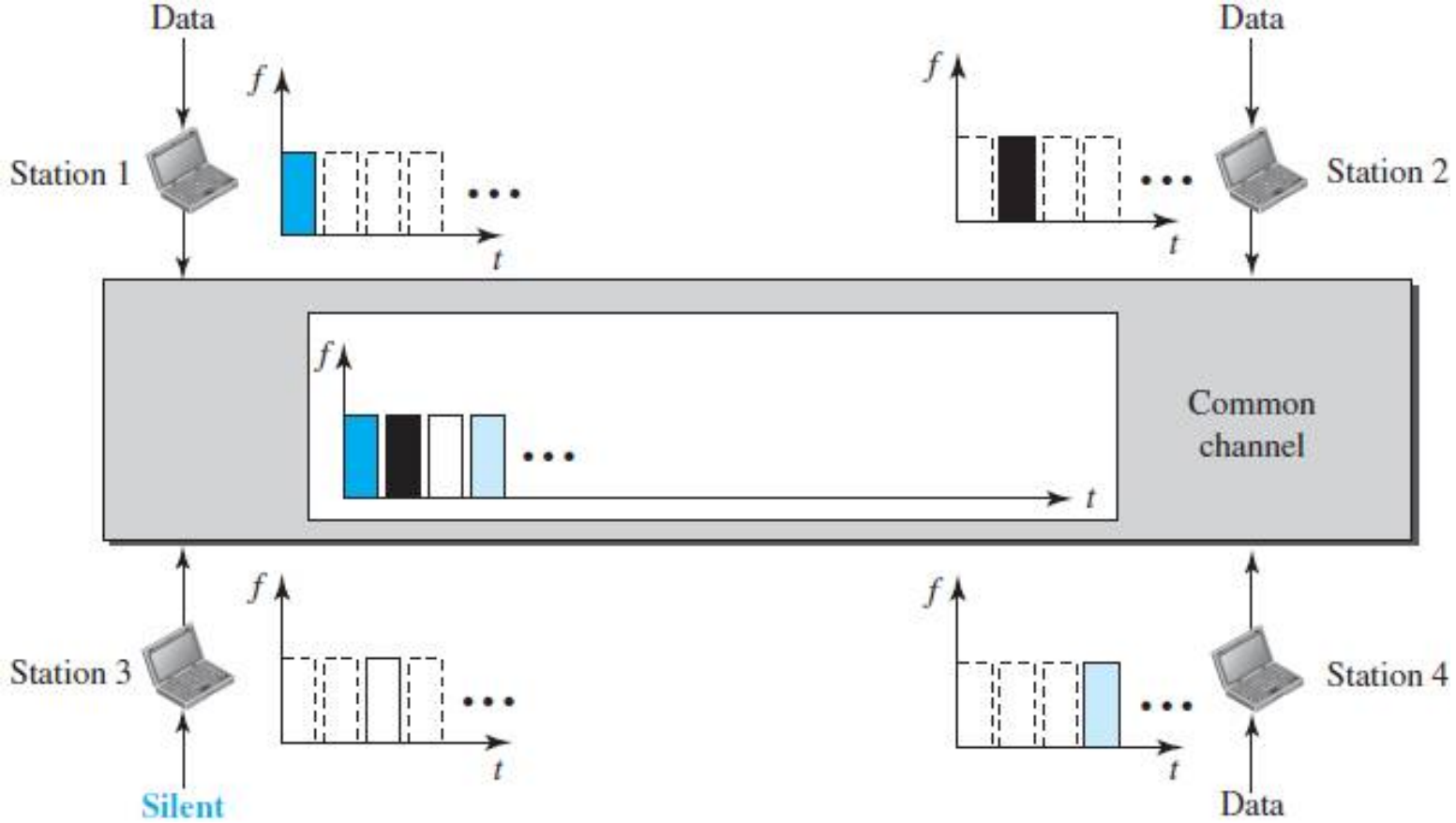
FDMA ...

- Specifies a predetermined frequency band for the entire period of communication
  - Stream data (a continuous flow of data that may not be packetized) can easily be used with FDMA

FDM	FDMA
A physical layer technique	An access method in the data-link layer
Combines the loads from low-bandwidth channels and transmits them by using a high-bandwidth channel	The data-link layer in each station tells its physical layer to make a bandpass signal from the data passed to it
Channels that are combined are low-pass	
Multiplexer modulates the signals, combines them, and creates a bandpass signal	No physical multiplexer at the physical layer The signals created at each station are automatically bandpass-filtered They are mixed when they are sent to the common channel
bandwidth of each channel is shifted by the multiplexer	

Time-division multiple access (TDMA)

- **Stations share the bandwidth of the** channel in time
- Each station is allocated a time slot during which it can send data



TDMA

- Problem: achieving synchronization between the different stations
- Each station needs to know the beginning of its slot and the location of its slot - may be difficult because of propagation delays introduced in the system if the stations are spread over a large area
- To compensate for the delays - -> insert *guard times*
- *Synchronization is normally accomplished by having some synchronization bits* (normally referred to as *preamble bits) at the beginning of each slot*

TDM	TDMA
Aphysical layer technique that combines the data from slower channels and transmits them by using a faster channel	An access method in the data-link layer data-link layer in each station tells its physical layer to use the allocated time slot
Uses a physical multiplexer that interleaves data units from each channel	No physical multiplexer at the physical layer

Code-division multiple access (CDMA)

- **Was conceived several decades ago but recent** advances in electronic technology have finally made its implementation possible
- **One channel carries all transmissions simultaneously**
  - CDMA differs from FDMA in that only one channel occupies the entire bandwidth of the link
  - It differs from TDMA in that all stations can send data simultaneously; there is no timesharing

CDMA - *Analogy*

- CDMA simply means communication with different codes
- For example, in a large room with many people
  - Two people can talk privately in English if nobody else understands English
  - Another two people can talk in Chinese if they are the only ones who understand Chinese
  - and so on
- The common channel, the space of the room in this case, can easily allow communication between several couples, but in different languages (codes)

END