Module-2

Data Link Layer-Error Detection and Correction: Introduction, Block Coding, Cyclic Codes.
Data Link Control-DLC Services: Framing, Flow Control, Error Control, Connectionless and Connection Oriented, Data link layer protocols, High Level Data Link Control.
Media Access Control: Random Access, Controlled Access. Check Sum and Point to Point Protocol.

DATA LINK LAYER-ERROR DETECTION AND CORRECTION:

2.1 Introduction

2.1.1 Types of Errors

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

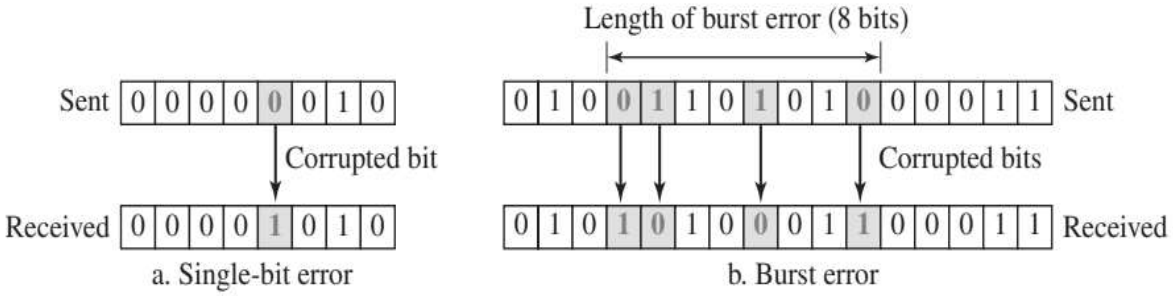

Fig. 2.1: Single-bit and burst error

A burst error is more likely to occur than a single-bit error because the duration of the noise signal is normally longer than the duration of 1 bit, which means that when noise affects data, it affects a set of bits. The number of bits affected depends on the data rate and duration of noise. For

example, if we are sending data at 1 kbps, a noise of 1/100 second can affect 10 bits; if we are sending data at 1 Mbps, the same noise can affect 10,000 bits.

2.1.2 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.

2.1.3 Detection versus Correction

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

2.1.4 Coding

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

2.2 Block Coding

In block coding, we divide our message into blocks, each of k bits, called data words. We add r redundant bits to each block to make the length n = k + r. The resulting n-bit blocks are called code words. How the extra r bits are chosen or calculated is something we will discuss later. For the moment, it is important to know that we have a set of data words, each of size k, and a set of

codewords, each of size of n. With k bits, we can create a combination of 2k data words; with n bits, we can create a combination of 2n codewords. Since n > k, the number of possible codewords is larger than the number of possible data words. The block coding process is one-to-one; the same data word is always encoded as the same codeword. This means that we have 2n − 2k codewords that are not used. We call these codewords invalid or illegal. The trick in error detection is the existence of these invalid codes, as we discuss next. If the receiver receives an invalid codeword, this indicates that the data was corrupted during transmission.

2.2.1 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.

Below Figure shows the role of block coding in error detection. The sender creates codewords out of data words 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 data word 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.

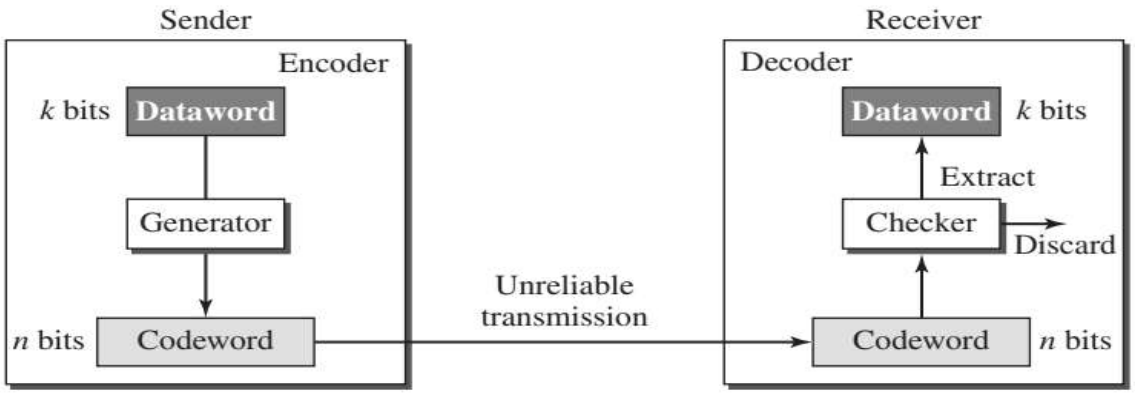

Fig. 2.2: Process of error detection in block coding

Hamming Distance

One of the central concepts in coding for error control is the idea of the Hamming distance. The Hamming distance between two words (of the same size) is the number of differences between the corresponding bits. We show the Hamming distance between two words x and y as d(x, y). We may wonder why Hamming distance is important for error detection. The reason is that the Hamming distance between the received codeword and the sent codeword is the number of bits that are corrupted during transmission. For example, if the codeword 00000 is sent and 01101 is received, 3 bits are in error and the Hamming distance between the two is d(00000, 01101) = 3. In other words, if the Hamming distance between the sent and the received codeword is not zero, the codeword has been corrupted during transmission.

The Hamming distance can easily be found if we apply the XOR operation ($\oplus$) on the two words and count the number of 1s in the result. Note that the Hamming distance is a value greater than or equal to zero.

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. Now let us find the minimum Hamming distance in a code if we want to be able to detect up to s errors. If s errors occur during transmission, the Hamming distance between the sent codeword and received codeword is s. 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. In other words, if the minimum distance between all valid codewords is (s + 1), the received codeword cannot be erroneously mistaken for another codeword. The error will be detected. We need to clarify a point here: Although a code with $d_{min} = s + 1$ may be able to detect more than s errors in some special cases, only s or fewer errors are guaranteed to be detected.

To guarantee the detection of up to s errors in all cases, the minimum Hamming distance in a block code must be $d_{min} = s + 1$.

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 Below Figure. This means that d_{min} must be an integer greater than s or $d_{min} = s + 1$.

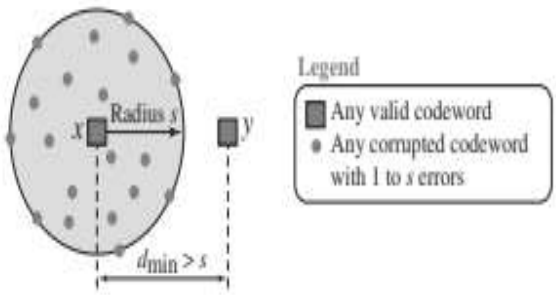

Fig. 2.2.1: Geometric concept explaining d_{min} in error detection

Linear Block Codes

Almost all block codes used today belong to a subset of block codes called linear block codes. The use of nonlinear block codes for error detection and correction is not as widespread because their structure makes theoretical analysis and implementation difficult. We therefore concentrate on linear block codes. The formal definition of linear block codes requires the knowledge of abstract algebra (particularly Galois fields), which is beyond the scope of this book. We therefore give an informal definition. For our purposes, a linear block code is a code in which the exclusive OR (addition modulo-2) of two valid codewords creates another valid codeword.

Parity-Check Code

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

Dataword	*Codeword*	*Dataword*	*Codeword*
0000	**0000**0	1000	**1000**1
0001	**0001**1	1001	**1001**0
0010	**0010**1	1010	**1010**0
0011	**0011**0	1011	**1011**1
0100	**0100**1	1100	**1100**0
0101	**0101**0	1101	**1101**1
0110	**0110**0	1110	**1110**1
0111	**0111**1	1111	**1111**0

Table 2.1 Simple parity-check code C(5, 4)

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

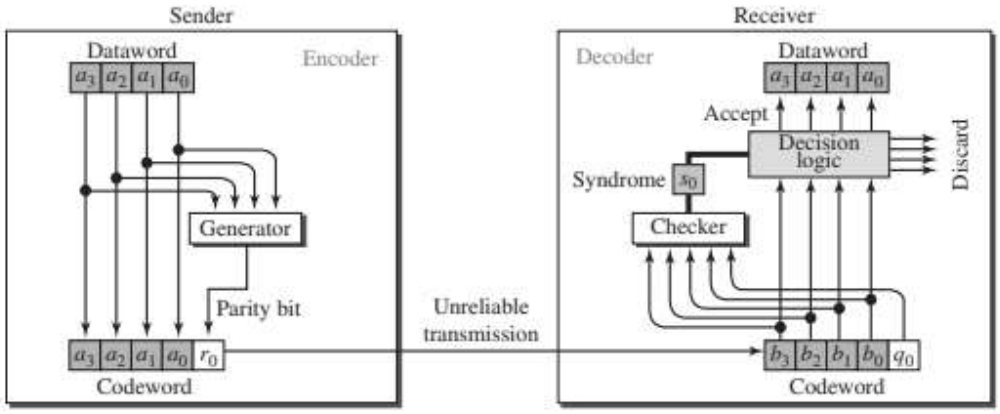

Fig. 2.2.2: Encoder and decoder for simple parity-check code

The encoder uses a generator that takes a copy of a 4-bit data word (a_0, a_1, a_2, and a_3) and generates a parity bit r_0. The data word 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 data word; the result is the parity bit. In other words,

$$r_0 = a_3 + a_2 + a_1 + a_0$$

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. 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. 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.

$$s_0 = b_3 + b_2 + b_1 + b_0 + q_0$$

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 code word is accepted as the data word; if the syndrome is 1, the data portion of the received codeword is discarded. The data word is not created.

2.3 Cyclic Codes

Cyclic codes are special linear block codes with one extra property. In a cyclic code, if a codeword is cyclically shifted (rotated), the result is another codeword. For example, if 1011000 is a codeword and we cyclically left-shift, then 0110001 is also a codeword. In this case, if we call the bits in the first word a_0 to a_6, and the bits in the second word b_0 to b_6, we can shift the bits by using the following:

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

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

2.3.1 Cyclic Redundancy Check (CRC)

Subset of cyclic codes called the cyclic redundancy check (CRC), which is used in networks such as LANs and WANs.

Below Table shows an example of a CRC code. We can see both the linear and cyclic properties of this code.

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

Table 2.2: A CRC code with C(7, 4)

Below Figure shows one possible design for the encoder and decoder.

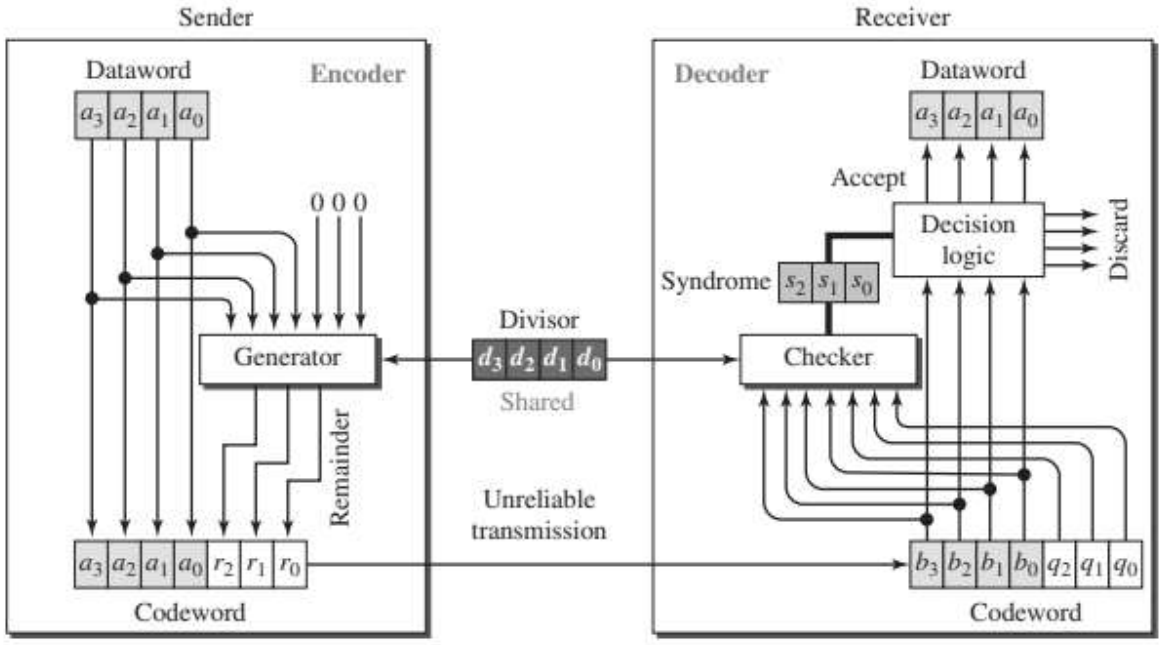

Fig. 2.3: CRC encoder and decoder

In the encoder, the data word has k bits (4 here); the codeword has n bits (7 here). The size of the data word is augmented by adding n − k (3 here) 0s to the right-hand side of the word. The n-bit result is fed into the generator. The generator uses a divisor of size n − k + 1 (4 here), predefined and agreed upon. The generator divides the augmented data word by the divisor. The quotient of the division is discarded; the remainder ($r_2r_1r_0$) is appended to the data word 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 left most bits of the codeword are accepted as the data word (interpreted as no error); otherwise, the 4 bits are discarded (error).

Encoder

The encoder takes a data word and augments it with n − k number of 0s. It then divides the augmented data word by the divisor, as shown in below Figure.

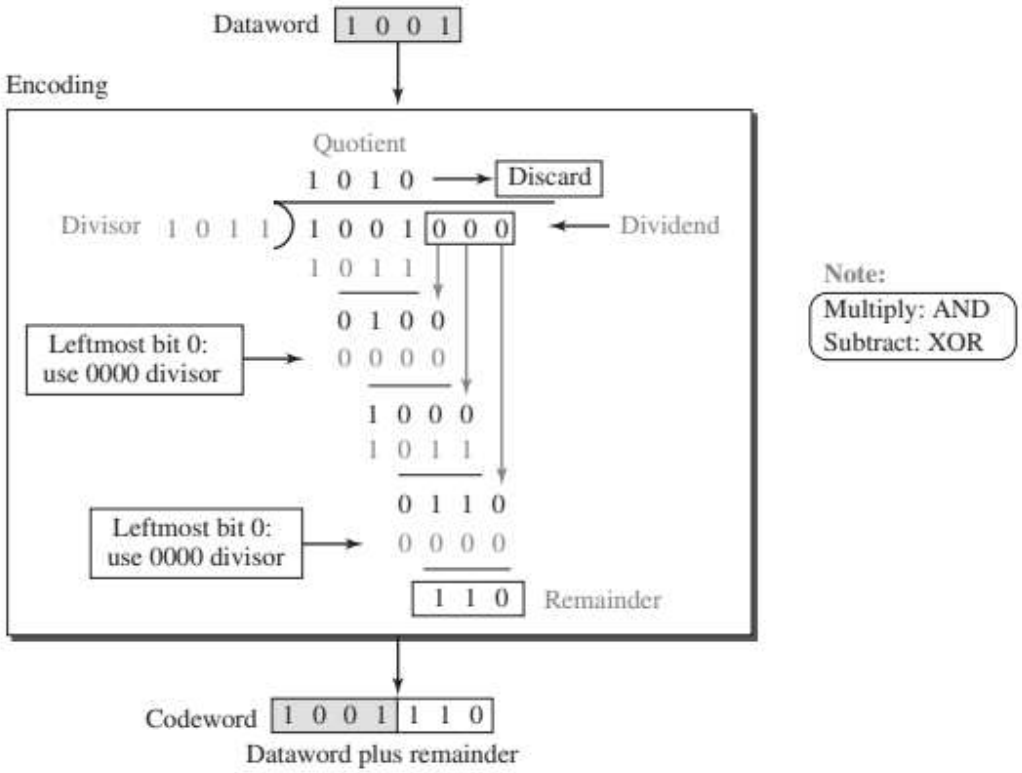

Fig. 2.3.1: Division in CRC encoder

The process of binary division is the same as the familiar division process we use for decimal numbers. However, addition and subtraction in this case are the same; we use the XOR operation to do both.

As in decimal division, the process is done step by step. In each step, a copy of the divisor is XORed with the 4 bits of the dividend. The result of the XOR operation (remainder) is 3 bits (in this case), which is used for the next step after 1 extra bit is pulled down to make it 4 bits long. There is one important point we need to remember in this type of division. If the leftmost bit of the dividend (or the part used in each step) is 0, the step cannot use the regular divisor; we need to use an all-0s divisor.

When there are no bits left to pull down, we have a result. The 3-bit remainder forms the **check bits** (r_2, r_1, and r_0). They are appended to the data word to create the codeword.

Decoder

The codeword can change during transmission. The decoder does the same division process as the encoder. The remainder of the division is the syndrome. If the syndrome is all 0s, there is no error with a high probability; the data word is separated from the received codeword and accepted. Otherwise, everything is discarded. Below Figure shows two cases: The left-hand figure shows the value of the syndrome when no error has occurred; the syndrome is 000. The right-hand part of the figure shows the case in which there is a single error. The syndrome is not all 0s (it is 011)

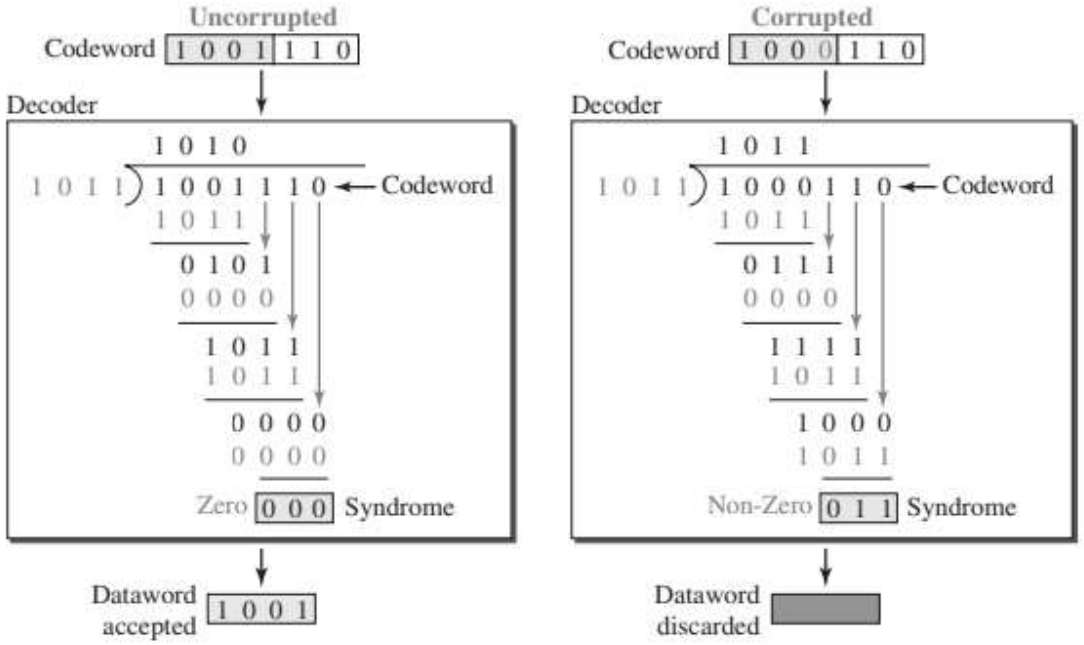

Fig. 2.3.2: Division in the CRC decoder for two cases

Divisor

How the divisor 1011 is chosen. This depends on the expectation we have from the code.

2.3.2 Polynomials

A better way to understand cyclic codes and how they can be analyzed is to represent them as polynomials. Again, this section is optional.

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. Below Figure shows a binary pattern and its polynomial representation. In Below Figure (a) we show how to translate a binary pattern into a polynomial; in below Figure (b) 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.

Below Figure shows one immediate benefit; a 7-bit pattern can be replaced by three terms. The benefit is even more conspicuous when we have a polynomial such as

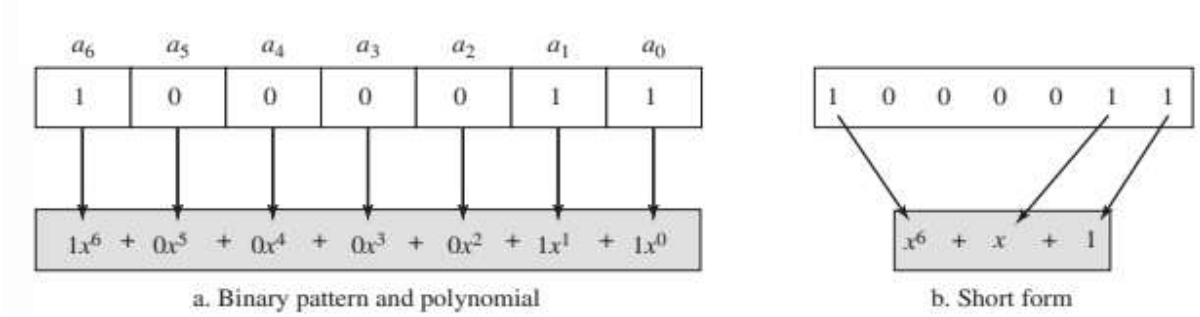

Fig. 2.3.3: A polynomial to represent a binary word

$x^{23} + x^3 + 1$. Here the bit pattern is 24 bits in length (three 1s and twenty-one 0s) while the polynomial is just three terms.

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. However, note that if we add, for example, three polynomials and we get x^2 three times, we delete a pair of them and keep the third.

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.

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:

$$\begin{aligned}(x^5+x^3+x^2+x)(x^2+x+1) &= x^7+x^6+x^5+x^5+x^4+x^3+x^4+x^3+x^2+x^3+x^2+x \\ &= x^7+x^6+x^3+x\end{aligned}$$

Dividing One Polynomial by Another

Division of polynomials is conceptually the same as the binary division we discussed for an encoder. 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. Note that we do not have negative powers in the polynomial representation.

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

When we augmented the data word in the encoder of Figure 2.6, we actually shifted the bits to the left. Also note that when we concatenate two bit patterns, we shift the first polynomial to the left and then add the second polynomial.

2.3.3 Cyclic Code Encoder Using Polynomials

We show the creation of a code word from a data word. Below Figure is the polynomial version of Figure2.6. We can see that the process is shorter. The data word 1001 is represented as $x^3 + 1$. The divisor 1011 is represented as $x^3 + x + 1$. To find the augmented data word, we have left-shifted the data word 3 bits (multiplying by x^3). 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 x^3 by the divisor and subtract (according to our previous definition of subtraction) 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.

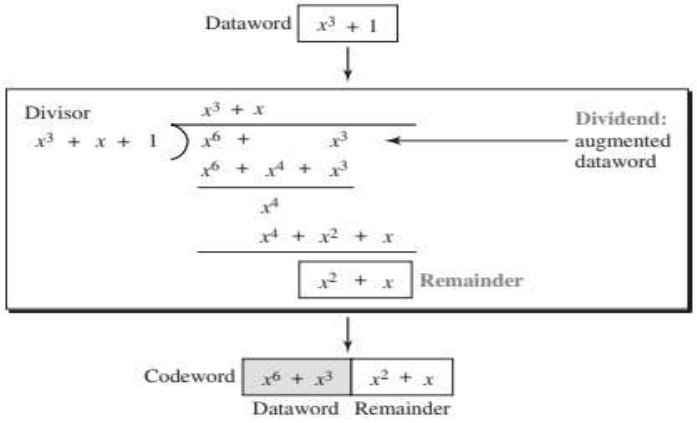

Fig. 2.3.4: CRC division using polynomials

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. (Of course, one could argue that the all-0s divisor step can also be eliminated in binary division.) In a polynomial representation, the divisor is normally referred to as the generator polynomial t(x).

2.3.4 Cyclic Code Analysis

We can analyze a cyclic code to find its capabilities by using polynomials. We define the following, where f(x) is a polynomial with binary coefficients.

Dataword: d(x) Codeword: c(x) Generator: g(x) Syndrome: s(x) Error: e(x)

If s(x) is not zero, then one or more bits is corrupted. However, if s(x) is zero, either no bit is corrupted or the decoder failed to detect any errors. (Note that ¦ means divide).

In a cyclic code,

1. If $s(x) \neq 0$, one or more bits is corrupted.
2. If $s(x) = 0$, either
   a. No bit is corrupted, or
   b. Some bits are corrupted, but the decoder failed to detect them.

In our analysis we want to find the criteria that must be imposed on the generator, g(x) to detect the type of error we especially want to be detected. Let us first find the relationship among the sent codeword, error, received codeword, and the generator. We can say

Received codeword = c(x) + e(x)

In other words, the received codeword is the sum of the sent codeword and the error. The receiver divides the received codeword by g(x) to get the syndrome. We can write this as

$$\frac{\text{Received codeword}}{g(x)} = \frac{c(x)}{g(x)} + \frac{e(x)}{g(x)}$$

The first term at the right-hand side of the equality has a remainder of zero (according to the definition of codeword). So the syndrome is actually the remainder of the second term on the right-hand side. If this term does not have a remainder (syndrome = 0), either e(x) is 0 or e(x) is divisible

by g(x). We do not have to worry about the first case (there is no error); the second case is very important. Those errors that are divisible by g(x) are not caught.

Single-Bit Error

What should the structure of g(x) be to guarantee the detection of a single-bit error? A single-bit error is e(x) = xi , where i is the position of the bit. If a single-bit error is caught, then xi is not divisible by g(x). (Note that when we say not divisible, we mean that there is a remainder.) If g(x) has at least two terms (which is normally the case) and the coefficient of x0 is not zero (the rightmost bit is 1), then e(x) cannot be divided by g(x).

Standard Polynomials

Some standard polynomials used by popular protocols for CRC generation are shown in below Table along with the corresponding bit pattern.

Name	*Polynomial*	*Used in*
CRC-8	$x^8 + x^2 + x + 1$ **100000111**	ATM header
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$ **11000110101**	ATM AAL
CRC-16	$x^{16} + x^{12} + x^5 + 1$ **1000100000100001**	HDLC
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ **10000010011000001000111011011 0111**	LANs

Table 2.3: Standard polynomial

2.3.5 Advantages of Cyclic Codes

We have seen that cyclic codes have a very good performance in detecting single-bit errors, double errors, an odd number of errors, and burst errors. They can easily be implemented in hardware and software. They are especially fast when implemented in hardware. This has made cyclic codes a good candidate for many networks.

2.3.6 Other Cyclic Codes

The cyclic codes we have discussed in this section are very simple. The check bits and syndromes can be calculated by simple algebra. There are, however, more powerful polynomials that are based on abstract algebra involving Galois fields. These are beyond the scope of this book. One of the most interesting of these codes is the **Reed-Solomon code** used today for both detection and correction.

2.3.7 Hardware Implementation

One of the advantages of a cyclic code is that the encoder and decoder can easily and cheaply be implemented in hardware by using a handful of electronic devices. Also, a hardware implementation increases the rate of check bit and syndrome bit calculation. In this section, we try to show, step by step, the process.

Divisor

Let us first consider the divisor. We need to note the following points:

1. The divisor is repeatedly XORed with part of the dividend.
2. The divisor has $n - k + 1$ bits which either are predefined or are all 0s. In other words, the bits do not change from one dataword to another. In our previous example, the divisor bits were either 1011 or 0000. The choice was based on the leftmost bit of the part of the augmented data bits that are active in the XOR operation.
3. A close look shows that only $n - k$ bits of the divisor are needed in the XOR operation. The leftmost bit is not needed because the result of the operation is always 0, no matter what the value of this bit. The reason is that the inputs to this XOR operation are either both 0s or both 1s. In our previous example, only 3 bits, not 4, are actually used in the XOR operation.

Using these points, we can make a fixed (hardwired) divisor that can be used for a cyclic code if we know the divisor pattern. Below Figure shows such a design for our previous example. We have also shown the XOR devices used for the operation.

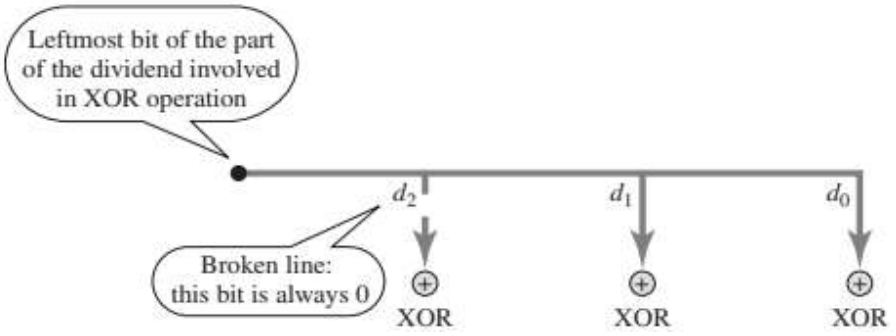

Fig. 2.3.5: Hardwired design of the divisor in crc

Augmented Dataword

In our paper-and-pencil division process in Figure 2.3.1, we show the augmented dataword as fixed in position with the divisor bits shifting to the right, 1 bit in each step. The divisor bits are aligned with the appropriate part of the augmented dataword. Now that our divisor is fixed, we need instead to shift the bits of the augmented dataword to the left (opposite direction) to align the divisor bits with the appropriate part. There is no need to store the augmented dataword bits.

Remainder

For example, the remainder is 3 bits (n − k bits in general) in length. We can use three registers (single-bit storage devices) to hold these bits. To find the final remainder of the division, we need to modify our division process. The following is the step-by-step process that can be used to simulate the division process in hardware (or even in software).

1. We assume that the remainder is originally all 0s (000 in our example).
2. At each time click (arrival of 1 bit from an augmented dataword), we repeat the following two actions:
    a. We use the leftmost bit to make a decision about the divisor (011 or 000).
    b. The other 2 bits of the remainder and the next bit from the augmented dataword (total of 3 bits) are XORed with the 3-bit divisor to create the next remainder.

Below Figure shows this simulator, but note that this is not the final design; there will be more improvements.

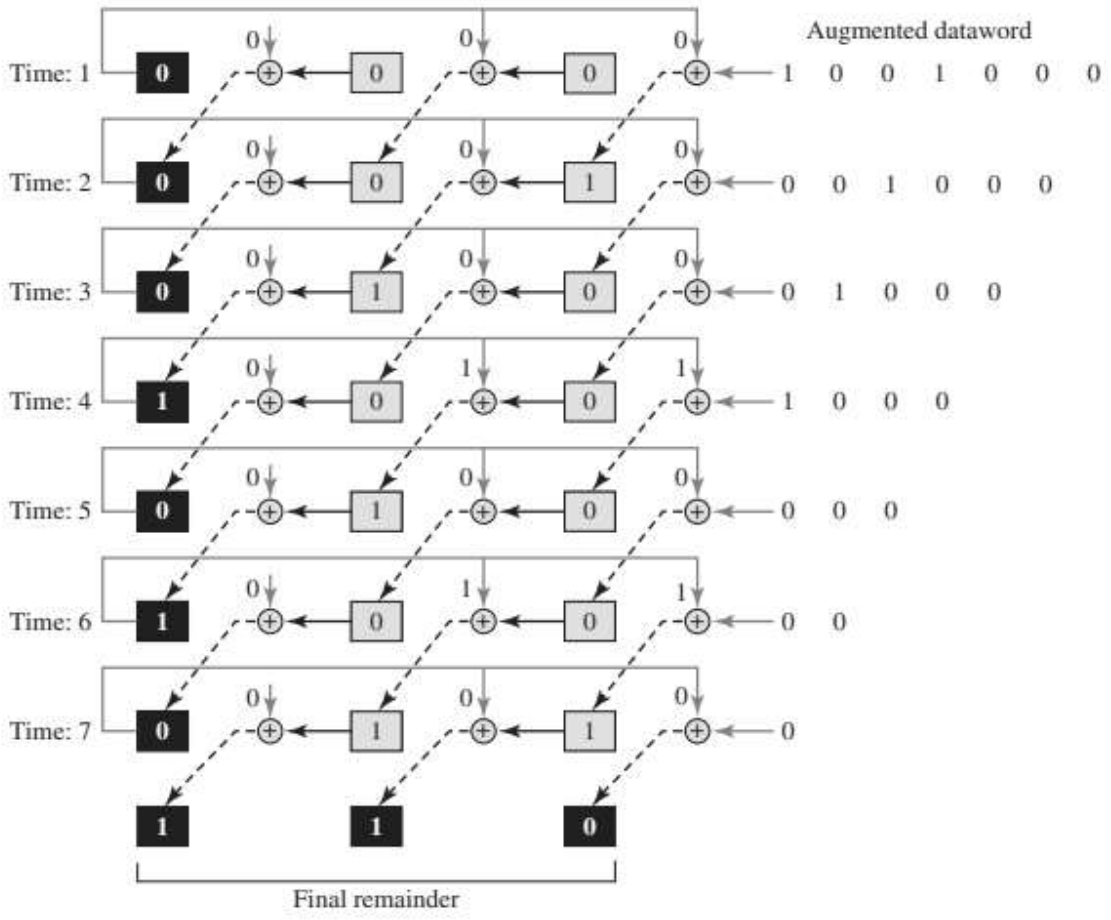

Fig. 2.3.6: Simulation of division in CRC encoder

At each clock tick, shown as different times, one of the bits from the augmented dataword is used in the XOR process. If we look carefully at the design, we have seven steps here, while in the paper-and-pencil method we had only four steps. The first three steps have been added here to make each step equal and to make the design for each step the same. Steps 1, 2, and 3 push the first 3 bits to the remainder registers; steps 4, 5, 6, and 7 match the paper-and-pencil design. Note that the values in the remainder register in steps 4 to 7 exactly match the values in the paper-and-pencil design. The final remainder is also the same.

The above design is for demonstration purposes only. It needs simplification to be practical. First, we do not need to keep the intermediate values of the remainder bits; we need only the final bits. We therefore need only 3 registers instead of 24. After the XOR operations, we do not need the bit values of the previous remainder. Also, we do not need 21 XOR devices; two are enough because

the output of an XOR operation in which one of the bits is 0 is simply the value of the other bit. This other bit can be used as the output. With these two modifications, the design becomes tremendously simpler and less expensive, as shown in below Figure.

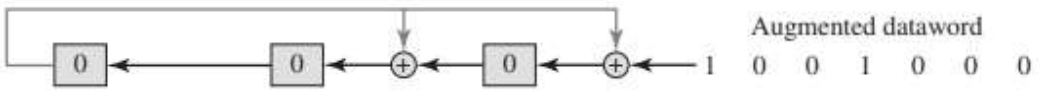

Fig. 2.3.7: The CRC encoder design using shift registers

We need, however, to make the registers shift registers. A 1-bit shift register holds a bit for a duration of one clock time. At a time click, the shift register accepts the bit at its input port, stores the new bit, and displays it on the output port. The content and the output remain the same until the next input arrives. When we connect several 1-bit shift registers together, it looks as if the contents of the register are shifting.

General Design

A general design for the encoder and decoder is shown in Figure 2.14.

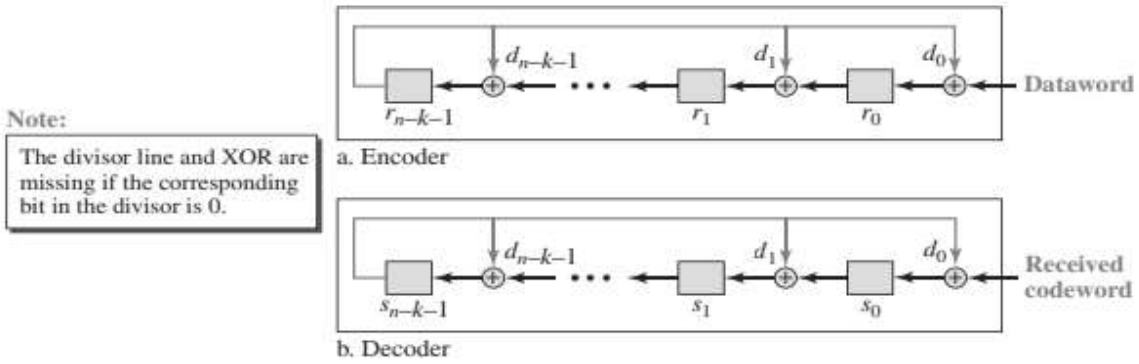

Fig. 2.3.8: General design of encoder and decoder of a CRC code

Note that we have n − k 1-bit shift registers in both the encoder and decoder. We have up to n − k XOR devices, but the divisors normally have several 0s in their pattern, which reduces the number of devices. Also note that, instead of augmented data words, we show the dataword itself as the input because after the bits in the dataword are all fed into the encoder, the extra bits, which all are 0s, do not have any effect on the rightmost XOR. Of course, the process needs to be continued for

another n − k steps before the check bits are ready. This fact is one of the criticisms of this design. Better schemes have been designed to eliminate this waiting time (the check bits are ready after k steps), but we leave this as a research topic for the reader. In the decoder, however, the entire codeword must be fed to the decoder before the syndrome is ready.

2.4 CHECKSUM

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). Note that in the real implementation, the checksum unit is not necessarily added at the end of the message; it can be inserted in the middle of the message.

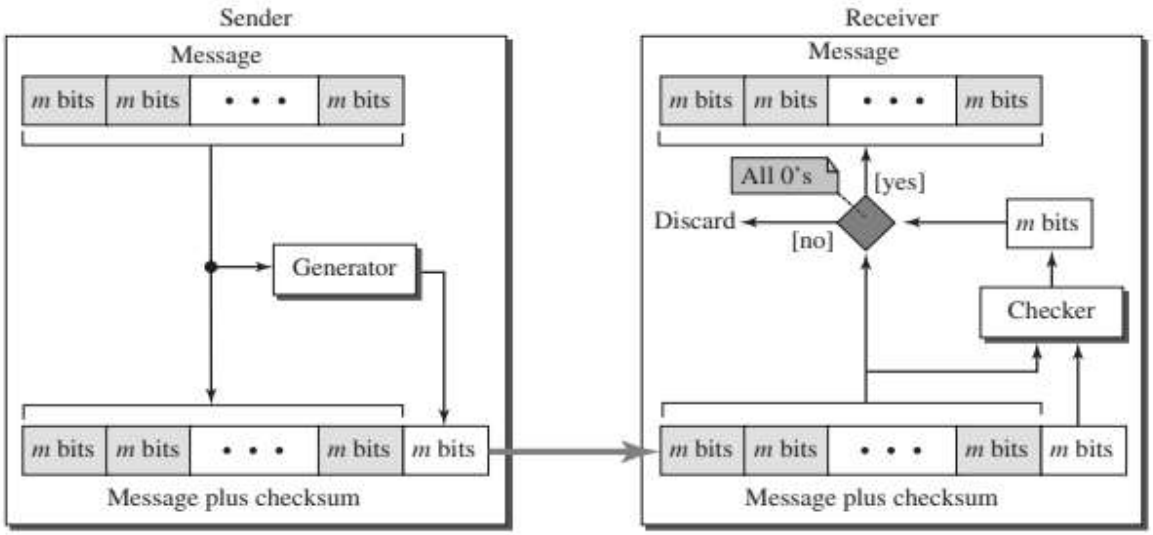

Fig. 2.4: Checksum

Procedure to calculate checksum

Sender	*Receiver*
1. The message is divided into 16-bit words.	1. The message and the checksum are received.
2. The value of the checksum word is initially set to zero.	2. The message is divided into 16-bit words.
3. All words including the checksum are added using one's complement addition.	3. All words are added using one's complement addition.
4. The sum is complemented and becomes the checksum.	4. The sum is complemented and becomes the new checksum.
5. The checksum is sent with the data.	5. If the value of the checksum is 0, the message is accepted; otherwise, it is rejected.

Algorithm

We can use the flow diagram of Figure 2.17 to show the algorithm for calculation of the checksum. A program in any language can easily be written based on the algorithm. Note that the first loop just calculates the sum of the data units in two's complement; the second loop wraps the extra bits created from the two's complement calculation to simulate the calculations in one's complement. This is needed because almost all computers today do calculation in two's complement.

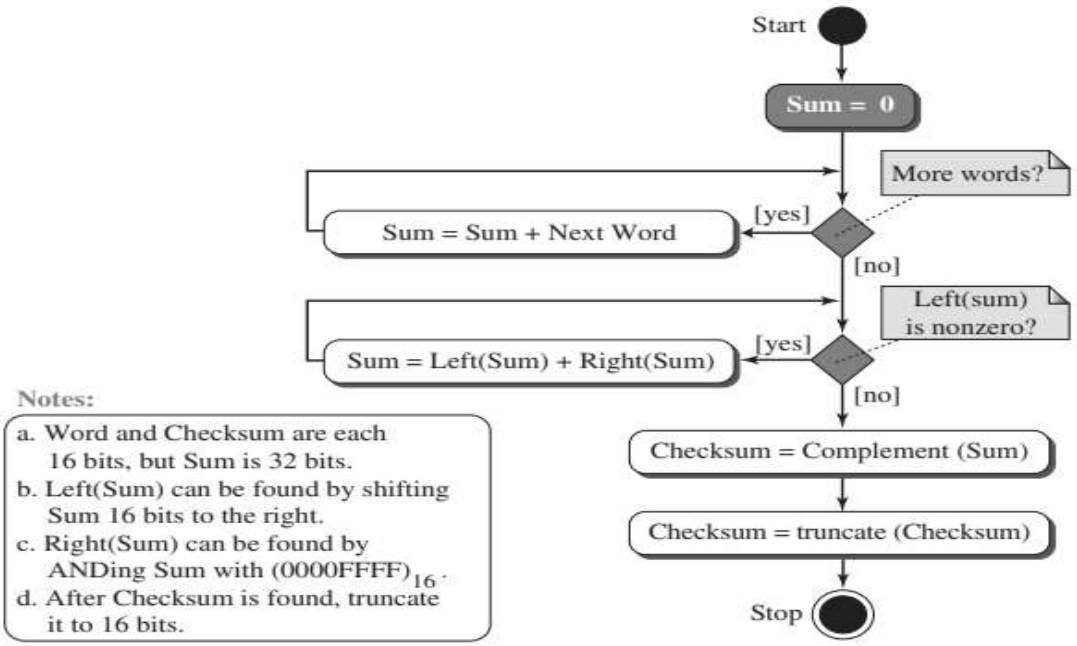

Fig. 2.4.1: Algorithm to calculate a traditional checksum

Performance

The traditional checksum uses a small number of bits (16) to detect errors in a message of any size (sometimes thousands of bits). However, it is not as strong as the CRC in error-checking capability. For example, if the value of one word is incremented and the value of another word is decremented by the same amount, the two errors cannot be detected because the sum and checksum remain the same. Also, if the values of several words are incremented but the sum and the checksum do not change, the errors are not detected. Fletcher and Adler have proposed some weighted checksums that eliminate the first problem. However, the tendency in the Internet, particularly in designing new protocols, is to replace the checksum with a CRC.

2.4.1 Other Approaches to the Checksum

As mentioned before, there is one major problem with the traditional checksum calculation. If two 16-bit items are transposed in transmission, the checksum cannot catch this error. The reason is that the traditional checksum is not weighted: it treats each data item equally. In other words, the order of data items is immaterial to the calculation. Several approaches have been used to prevent this problem. We mention two of them here: Fletcher and Adler.

Fletcher Checksum

The Fletcher checksum was devised to weight each data item according to its position. Fletcher has proposed two algorithms: 8-bit and 16-bit. The first, 8-bit Fletcher, calculates on 8-bit data items and creates a 16-bit checksum. The second, 16-bit Fletcher, calculates on 16-bit data items and creates a 32-bit checksum.

The 8-bit Fletcher is calculated over data octets (bytes) and creates a 16-bit checksum. The calculation is done modulo 256 (28), which means the intermediate results are divided by 256 and the remainder is kept. The algorithm uses two accumulators, L and R. The first simply adds data items together; the second adds a weight to the calculation. There are many variations of the 8-bit Fletcher algorithm; we show a simple one in Figure 2.4.2.

The 16-bit Fletcher checksum is similar to the 8-bit Fletcher checksum, but it is calculated over 16-bit data items and creates a 32-bit checksum. The calculation is done modulo 65,536.

Adler Checksum

The Adler checksum is a 32-bit checksum. Figure 2.4.3 shows a simple algorithm in flowchart form. It is similar to the 16-bit Fletcher with three differences. First, calculation is done on single bytes instead of 2 bytes at a time. Second, the modulus is a prime number (65,521) instead of 65,536. Third, L is initialized to 1 instead of 0. It has been proved that a prime modulo has a better detecting capability in some combinations of data.

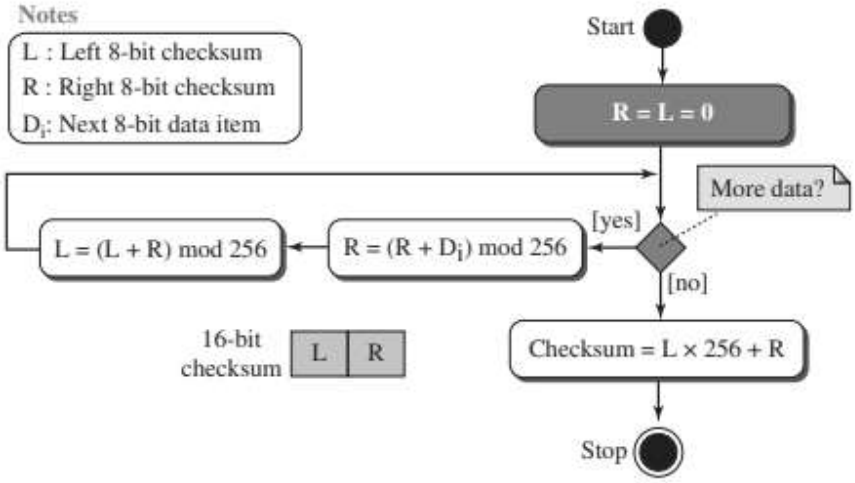

Fig. 2.4.2: Algorithm to calculate an 8-bit Fletcher checksum

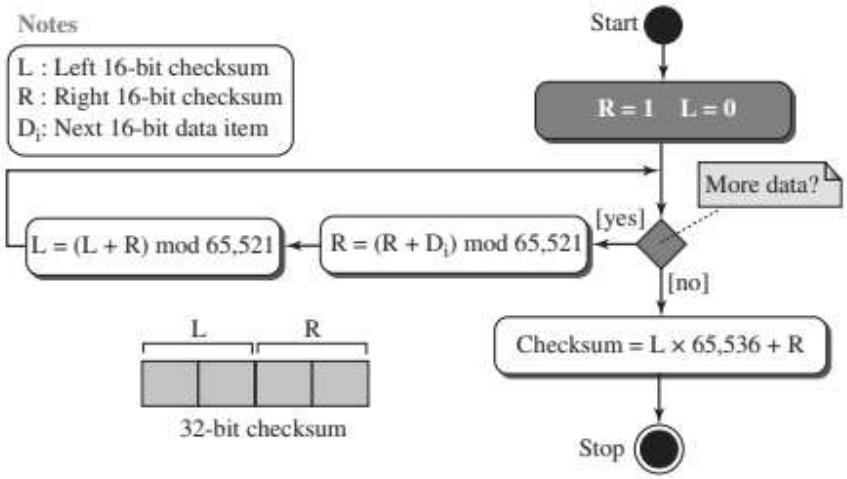

Fig. 2.4.3: Algorithm to calculate an Adler checksum

DATA LINK CONTROL-DLC SERVICES

The data link control (DLC) deals with procedures for communication between two adjacent nodes—node-to-node communication—no matter whether the link is dedicated or broadcast. Data link control functions include framing and flow and error control

2.5 Framing

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

The data-link layer, on the other hand, needs to pack bits into frames, so that each frame is distinguishable from another. Our postal system practices a type of framing. The simple act of inserting a letter into an envelope separates one piece of information from another; the envelope serves as the delimiter. In addition, each envelope defines the sender and receiver addresses, which is necessary since the postal system is a many-to-many carrier facility.

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.

Frame Size

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

In variable-size framing, we 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 approach and a bit-oriented approach.

Character-Oriented Framing

In character-oriented (or byte-oriented) framing, data to be carried are 8-bit characters from a coding system such as ASCII. The header, which normally carries the source and destination

addresses and other control information, and the trailer, which carries error detection redundant bits, are also multiples of 8 bits. To separate one frame from the next, an 8-bit (1-byte) flag is added at the beginning and the end of a frame. The flag, composed of protocol-dependent special characters, signals the start or end of a frame. Below Figure shows the format of a frame in a character-oriented protocol.

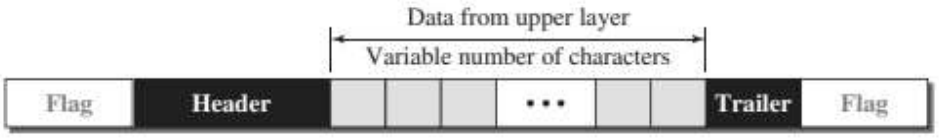

Fig. 2.5: A frame in a character-oriented protocol

Character-oriented framing was popular when only text was exchanged by the data-link layers. The flag could be selected to be any character not used for text communication. Now, however, we send other types of information such as graphs, audio, and video; any character used for the flag could also be part of the information. If this happens, the receiver, when it encounters this pattern in the middle of the data, thinks it has reached the end of the frame. To fix this problem, a byte-stuffing strategy was added to character-oriented framing. In byte stuffing (or character stuffing), a special byte is added to the data section of the frame when there is a character with the same pattern as the flag. The data section is stuffed with an extra byte. This byte is usually called the escape character (ESC) and has a predefined bit pattern. Whenever the receiver encounters the ESC character, it removes it from the data section and treats the next character as data, not as a delimiting flag. Below Figure shows the situation.

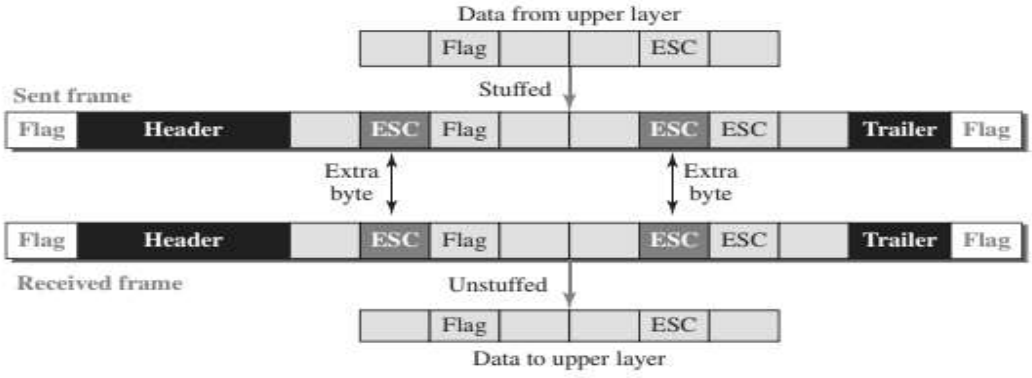

Fig. 2.5.1: Byte stuffing and unstuffing

Byte stuffing by the escape character allows the presence of the flag in the data section of the frame, but it creates another problem.

Character-oriented protocols present another problem in data communications. The universal coding systems in use today, such as Unicode, have 16-bit and 32-bit characters that conflict with 8-bit characters. We can say that, in general, the tendency is moving toward the bit-oriented protocols.

Bit-Oriented Framing In bit-oriented framing, the data section of a frame is a sequence of bits to be interpreted by the upper layer as text, graphic, audio, video, and so on. However, in addition to headers (and possible trailers), we still need a delimiter to separate one frame from the other. Most protocols use a special 8-bit pattern flag, 01111110, as the delimiter to define the beginning and the end of the frame, as shown in below Figure.

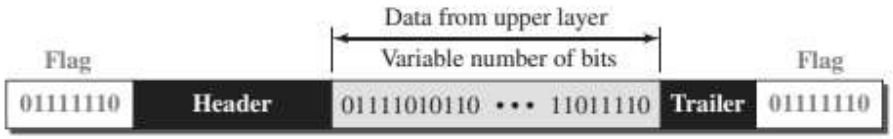

Fig. 2.5.2: A frame in a bit-oriented protocol

This flag can create the same type of problem we saw in the character-oriented protocols. That is, if the flag pattern appears in the data, we need to somehow inform the receiver that this is not the end of the frame. We do this by stuffing 1 single bit (instead of 1 byte) to prevent the pattern from looking like a flag. The strategy is called bit stuffing. In bit stuffing, if a 0 and five consecutive 1 bits are encountered, an extra 0 is added. This extra stuffed bit is eventually removed from the data by the receiver. 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.

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

This means that if the flaglike 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.

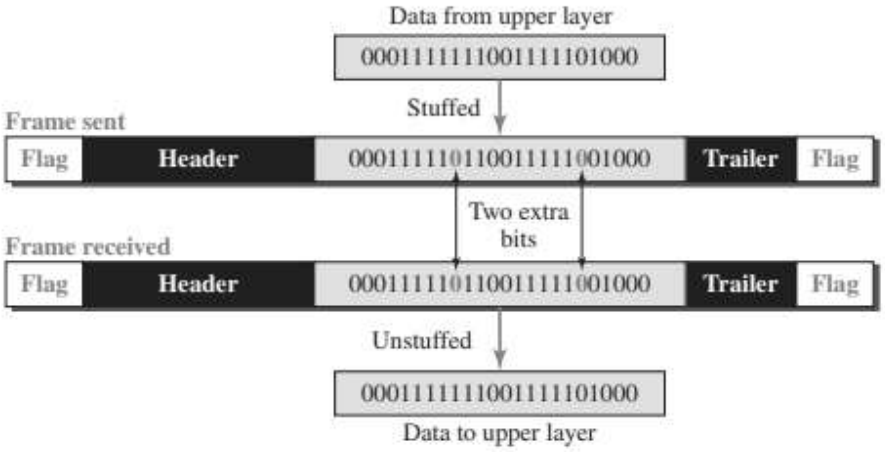

Fig. 2.5.3: Bit stuffing and unstuffing

2.6 Flow Control

Whenever an entity produces items and another entity consumes them, there should be a balance between production and consumption rates. If the items are produced faster than they can be consumed, the consumer can be overwhelmed and may need to discard some items. If the items are produced more slowly than they can be consumed, the consumer must wait, and the system becomes less efficient. Flow control is related to the first issue. We need to prevent losing the data items at the consumer site.

In communication at the data-link layer, we are dealing with four entities: network and data-link layers at the sending node and network and data-link layers at the receiving node. Although we can have a complex relationship with more than one producer and consumer. we ignore the relationships between networks and data-link layers and concentrate on the relationship between two data-link layers, as shown in below Figure.

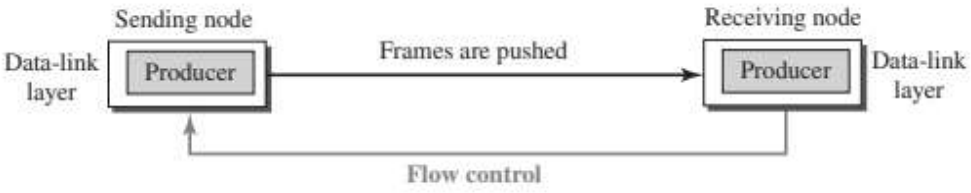

Fig. 2.6: Flow control at the data link layer

2.7 Error Control

Since the underlying technology at the physical layer is not fully reliable, we need to implement error control at the data-link layer to prevent the receiving node from delivering corrupted packets to its network layer. Error control at the data-link layer is normally very simple and implemented using one of the following two methods. In both methods, a CRC is added to the frame header by the sender and checked by the receiver.

- In the first method, if the frame is corrupted, it is silently discarded; if it is not corrupted, the packet is delivered to the network layer. This method is used mostly in wired LANs such as Ethernet.
- In the second method, if the frame is corrupted, it is silently discarded; if it is not corrupted, an acknowledgment is sent (for the purpose of both flow and error control) to the sender.

2.8 Connectionless and Connection Oriented

A DLC protocol can be either connectionless or connection-oriented.

Connectionless Protocol

In a connectionless protocol, frames are sent from one node to the next without any relationship between the frames; each frame is independent. Note that the term connectionless here does not mean that there is no physical connection (transmission medium) between the nodes; it means that there is no connection between frames. The frames are not numbered and there is no sense of ordering. Most of the data-link protocols for LANs are connectionless protocols.

Connection-Oriented Protocol

In a connection-oriented protocol, a logical connection should first be established between the two nodes (setup phase). After all frames that are somehow related to each other are transmitted (transfer phase), the logical connection is terminated (teardown phase). In this type of communication, the frames are numbered and sent in order. If they are not received in order, the receiver needs to wait until all frames belonging to the same set are received and then deliver them in order to the network layer. Connection-oriented protocols are rare in wired LANs, but we can see them in some point-to-point protocols, some wireless LANs, and some WANs.

2.9 Data link layer protocols

Traditionally four protocols have been defined for the data-link layer to deal with flow and error control: Simple, Stop-and-Wait, Go-Back-N, and Selective-Repeat. Although the first two protocols still are used at the data-link layer, the last two have disappeared. We therefore briefly discuss the first two protocols in this chapter, in which we need to understand some wired and wireless LANs.

The behavior of a data-link-layer protocol can be better shown as a finite state machine (FSM). An FSM is thought of as a machine with a finite number of states. The machine is always in one of the states until an event occurs. Each event is associated with two reactions: defining the list (possibly empty) of actions to be performed and determining the next state (which can be the same as the current state). One of the states must be defined as the initial state, the state in which the machine starts when it turns on. In below Figure, we show an example of a machine using FSM. We have used rounded-corner rectangles to show states, colored text to show events, and regular black text to show actions. A horizontal line is used to separate the event from the actions, although later we replace the horizontal line with a slash. The arrow shows the movement to the next state.

The figure shows a machine with three states. There are only three possible events and three possible actions. The machine starts in state I. If event 1 occurs, the machine performs actions 1 and 2 and moves to state II. When the machine is in state II, two events may occur. If event 1 occurs, the machine performs action 3 and remains in the same state, state II. If event 3 occurs, the machine performs no action, but move to state I.

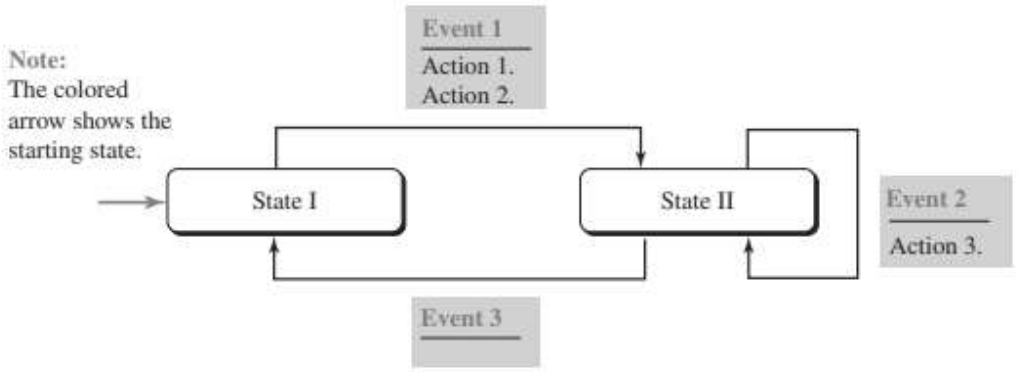

Fig. 2.9: Connectionless and connection-oriented service represented as FSMs

2.9.1 Simple Protocol

Our first protocol is a simple protocol with neither flow nor error control. We assume that the receiver can immediately handle any frame it receives. In other words, the receiver can never be overwhelmed with incoming frames. Below Figure shows the layout for this protocol.

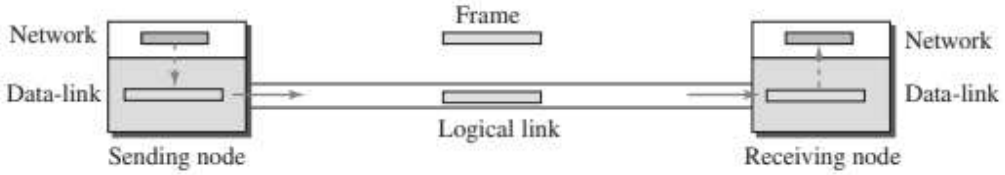

Fig. 2.9.1: Simple Protocol

The data-link layer at the sender gets a packet from its network layer, makes a frame out of it, and sends the frame. The data-link layer at the receiver receives a frame from the link, extracts the packet from the frame, and delivers the packet to its network layer. The data-link layers of the sender and receiver provide transmission services for their network layers.

FSMs

The sender site should not send a frame until its network layer has a message to send. The receiver site cannot deliver a message to its network layer until a frame arrives. We can show these requirements using two FSMs. Each FSM has only one state, the ready state. The sending machine remains in the ready state until a request comes from the process in the network layer. When this

event occurs, the sending machine encapsulates the message in a frame and sends it to the receiving machine. The receiving machine remains in the ready state until a frame arrives from the sending machine. When this event occurs, the receiving machine decapsulates the message out of the frame and delivers it to the process at the network layer. Below Figure shows the FSMs for the simple protocol.

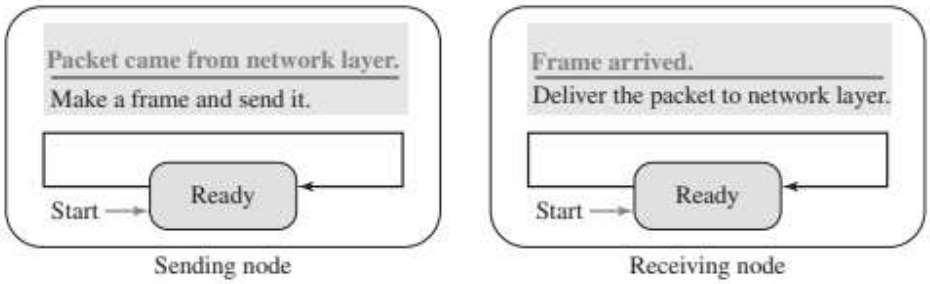

Fig. 2.9.2: FSMs for the simple protocol

2.9.2 Stop-and-Wait Protocol

Our second protocol is called the Stop-and-Wait protocol, which uses both flow and error control. We show a primitive version of this protocol here, but we discuss the more sophisticated version. when we have learned about sliding windows. In this protocol, the sender sends one frame at a time and waits for an acknowledgment before sending the next one. To detect corrupted frames, we need to add a CRC to each data frame. When a frame arrives at the receiver site, it is checked. 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. Below Figure shows the outline for the Stop-and-Wait protocol. Note that only one frame and one acknowledgment can be in the channels at any time.

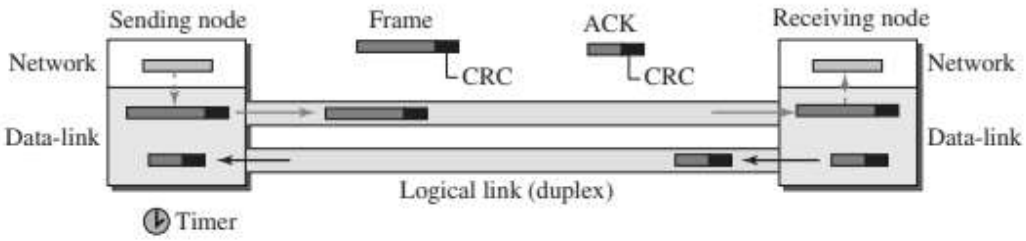

Fig. 2.9.3: Stop-and-Wait protocol

FSMs

Below Figure shows the FSMs for our primitive Stop-and-Wait protocol.

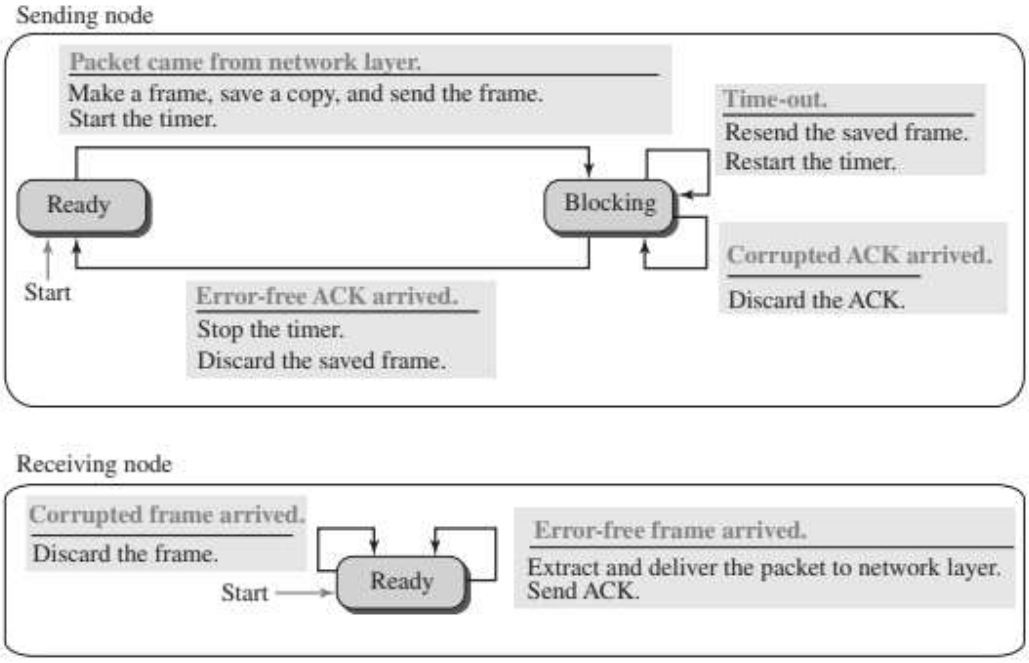

Fig. 2.9.4: FSM for the Stop-and-Wait protocol

We describe the sender and receiver states below.

Sender States

The sender is initially in the ready state, but it can move between the ready and blocking state.

- **Ready State.** When the sender is in this state, it is 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.** When the sender is in this 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.

Receiver

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.

Sequence and Acknowledgment Numbers

Duplicate packets, as much as corrupted packets, need to be avoided. As an example, assume we are ordering some item online. If each packet defines the specification of an item to be ordered, duplicate packets mean ordering an item more than once. we need to add **sequence numbers** to the data frames and **acknowledgment numbers** to the ACK frames. However, numbering in this case is very simple. Sequence numbers are 0, 1, 0, 1, 0, 1, . . . ; the acknowledgment numbers can also be 1, 0, 1, 0, 1, 0, … In other words, 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.

FSMs with Sequence and Acknowledgment Numbers

We can change the FSM in Figure 2.9.4 to include the sequence and acknowledgment numbers.

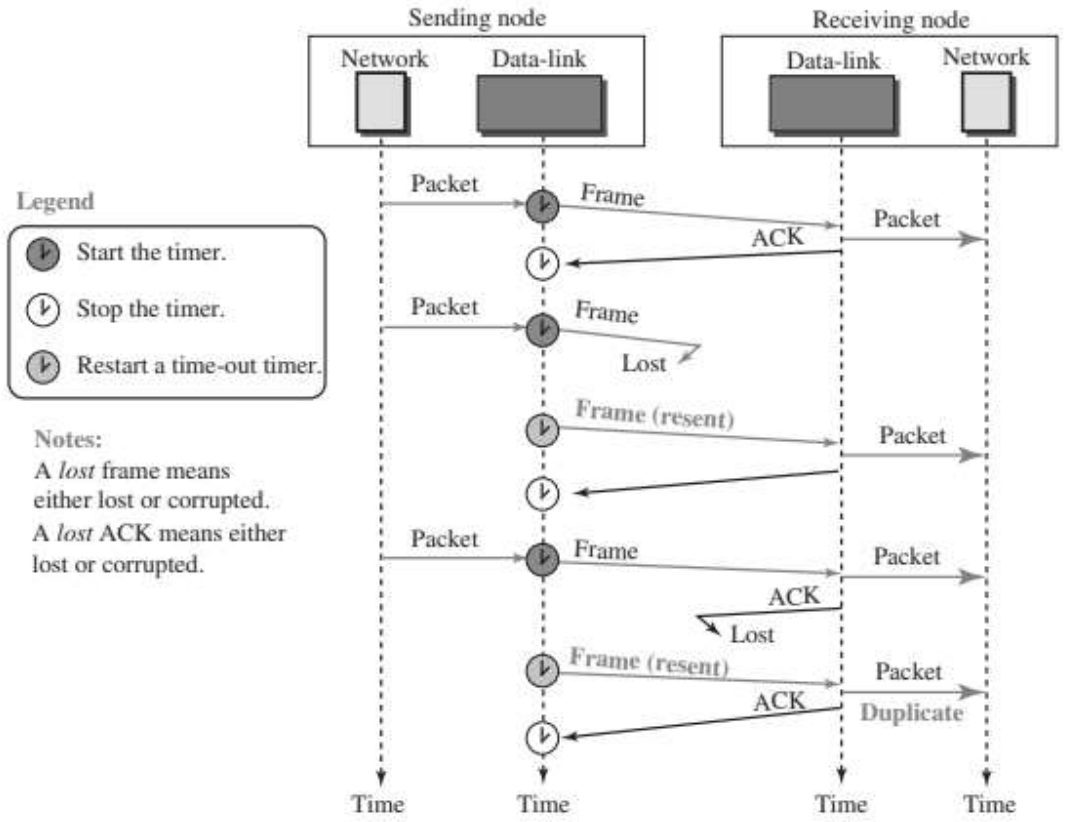

Fig. 2.9.5: Flow Diagram

2.9.3 Piggybacking

The two protocols we discussed in this section are designed for unidirectional communication, in which data is flowing only in one direction although the acknowledgment may travel in the other direction. Protocols have been designed in the past to allow data to flow in both directions. However, to make the communication more efficient, the data in one direction is piggybacked with the acknowledgment in the other direction. In other words, 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.

2.10 High level data link control (HDLC)

High-level Data Link Control (HDLC) is a bit-oriented protocol for communication over point-to-point and multipoint links. It implements the Stop-and-Wait protocol we discussed earlier.

Although this protocol is more a theoretical issue than practical, most of the concept defined in this protocol is the basis for other practical protocols such as PPP, which we discuss next, or the Ethernet protocol.

2.10.1 Configurations and Transfer Modes

HDLC provides two common transfer modes that can be used in different configurations: normal response mode (NRM) and asynchronous balanced mode (ABM). In normal response mode (NRM), the station configuration is unbalanced. We have one primary station and multiple secondary stations. A primary station can send commands; a secondary station can only respond. The NRM is used for both point-to-point and multipoint links, as shown in Figure 2.10.

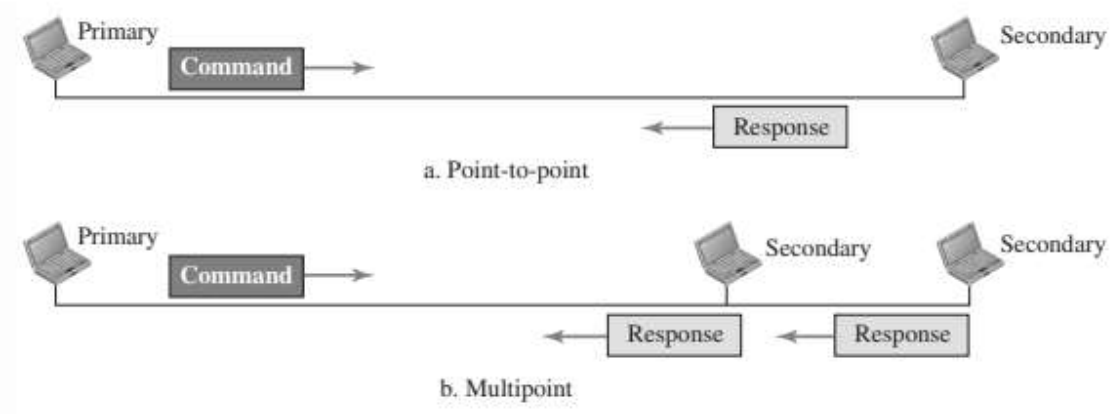

Fig. 2.10: Normal response mode

2.10.2 Framing

To provide the flexibility necessary to support all the options possible in the modes and configurations just described, HDLC defines three types of frames: information frames (I-frames), supervisory frames (S-frames), and unnumbered frames (U-frames). Each type of frame serves as an envelope for the transmission of a different type of message. I-frames are used to data-link user data and control information relating to user data (piggybacking). S-frames are used only to transport control information. U-frames are reserved for system management. Information carried by U-frames is intended for managing the link itself. Each frame in HDLC may contain up to six fields, as shown in Figure 2.10.2: a beginning flag field, an address field, a control field, an

information field, a frame check sequence (FCS) field, and an ending flag field. In multiple-frame transmissions, the ending flag of one frame can serve as the beginning flag of the next frame.

Fig. 2.10.1: Asynchronous balanced mode

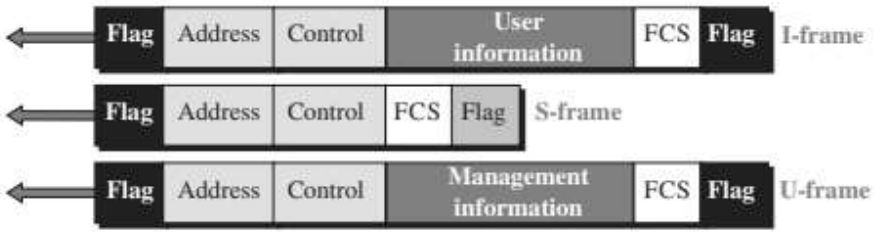

Fig. 2.10.2: HDLC Frames

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

The control field determines the type of frame and defines its functionality. The format is specific for the type of frame, as shown in below Figure.

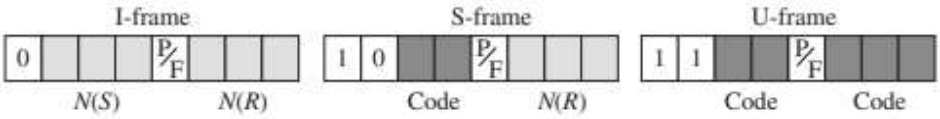

Fig. 2.10.3: Control field format for the different frame types

Control Field for I-Frames

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

Control Field for S-Frames

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

- **Receive ready (RR).** If the value of the code subfield is 00, it is an RR S-frame. This kind of frame acknowledges the receipt of a safe and sound frame or group of frames. In this case, the value of the N(R) field defines the acknowledgment number.
- **Receive not ready (RNR).** If the value of the code subfield is 10, it is an RNR S-frame. This kind of frame is an RR frame with additional functions. It acknowledges the receipt

of a frame or group of frames, and it announces that the receiver is busy and cannot receive more frames. It acts as a kind of congestion-control mechanism by asking the sender to slow down. The value of N(R) is the acknowledgment number.

- **Reject (REJ).** If the value of the code subfield is 01, it is an REJ S-frame. This is a NAK frame, but not like the one used for Selective Repeat ARQ. It is a NAK that can be used in Go-Back-N ARQ to improve the efficiency of the process by informing the sender, before the sender timer expires, that the last frame is lost or damaged. The value of N(R) is the negative acknowledgment number.
- **Selective reject (SREJ)**. If the value of the code subfield is 11, it is an SREJ S-frame. This is a NAK frame used in Selective Repeat ARQ. Note that the HDLC Protocol uses the term selective reject instead of selective repeat. The value of N(R) is the negative acknowledgment number.

Control Field for U-Frames

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

2.11 Point to point protocol (PPP)

One of the most common protocols for point-to-point access is the Point-to-Point Protocol (PPP). Today, millions of Internet users who need to connect their home computers to the server of an Internet service provider use PPP. The majority of these users have a traditional modem; they are 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 by far the most common.

2.11.1 Services

The designers of PPP have included several services to make it suitable for a point-to-point protocol, but have ignored some traditional services to make it simple.

Services Provided by PPP

PPP defines the format of the frame to be exchanged between devices. It also defines how two devices can negotiate the establishment of the link and the exchange of data. PPP is designed to accept payloads from several network layers (not only IP). Authentication is also provided in the protocol, but it is optional. The new version of PPP, called Multilink PPP, provides connections over multiple links. One interesting feature of PPP is that it provides network address configuration. This is particularly useful when a home user needs a temporary network address to connect to the Internet.

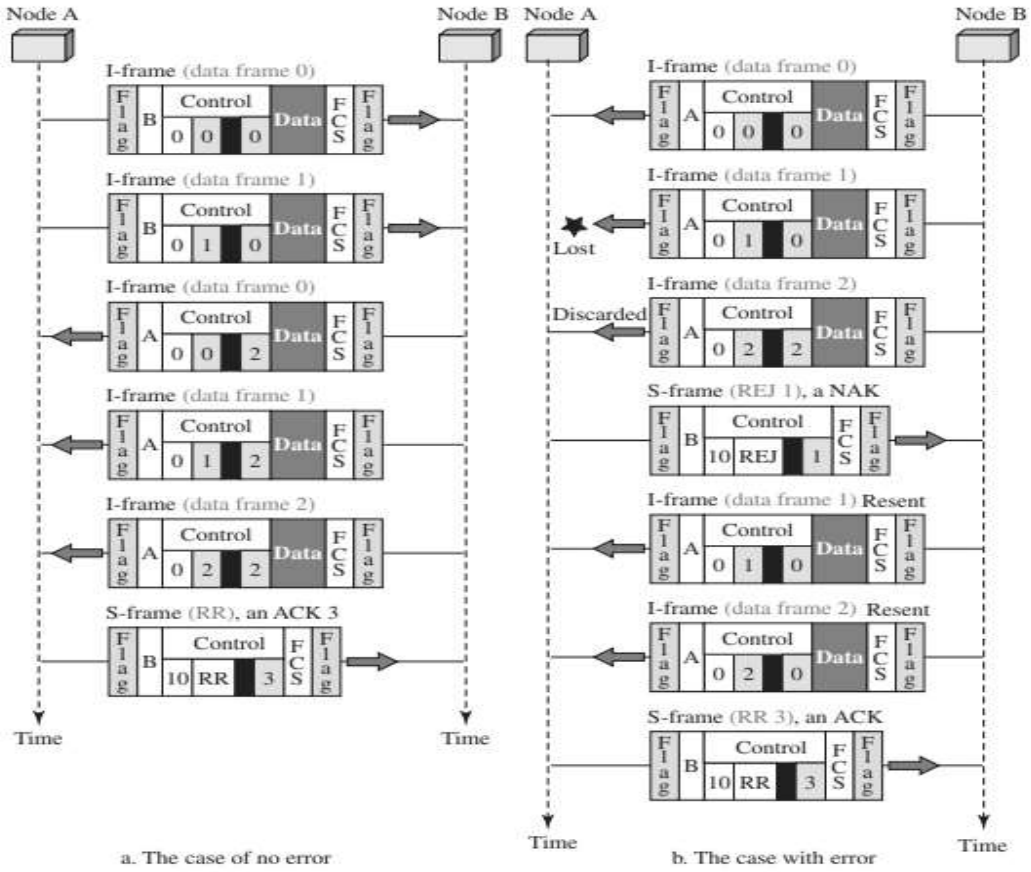

Fig. 2.11: Example of piggybacking with and without error

Services Not Provided by PPP

PPP does not provide flow control. A sender can send several frames one after another with no concern about overwhelming the receiver. PPP has 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. PPP does not provide a sophisticated addressing mechanism to handle frames in a multipoint configuration.

2.11.2 Framing

PPP uses a character-oriented (or byte-oriented) frame. below Figure shows the format of a PPP frame. The description of each field follows:

- **Flag**. A PPP frame starts and ends with a 1-byte flag with the bit pattern 01111110.
- **Address.** The address field in this protocol is a constant value and set to 11111111 (broadcast address).
- **Control.** This field is set to the constant value 00000011 (imitating unnumbered frames in HDLC). As we will discuss later, PPP does not provide any flow control. Error control is also limited to error detection.
- **Protocol.** The protocol field defines what is being carried in the data field: either user data or other information. This field is by default 2 bytes long, but the two parties can agree to use only 1 byte.
- **Payload field**. This field carries either the user data or other information that we will discuss shortly. The data field is a sequence of bytes with the default of a maximum of 1500 bytes; but this can be changed during negotiation. The data field is byte-stuffed if the flag byte pattern appears in this field. Because there is no field defining the size of the data field, padding is needed if the size is less than the maximum default value or the maximum negotiated value.
- **FCS**. The frame check sequence (FCS) is simply a 2-byte or 4-byte standard CRC.

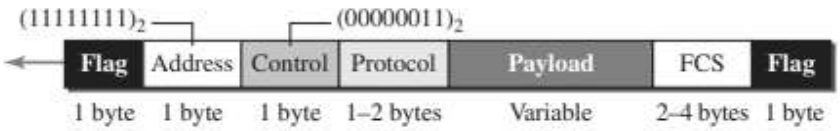

Fig. 2.11.1: PPP frame format

Byte Stuffing

Since PPP is a byte-oriented protocol, the flag in PPP is a byte that needs to be escaped whenever it appears in the data section of the frame. The escape byte is 01111101, which means that every time the flaglike pattern appears in the data, this extra byte is stuffed to tell the receiver that the next byte is not a flag. Obviously, the escape byte itself should be stuffed with another escape byte.

2.11.3 Transition Phases

A PPP connection goes through phases which can be shown in a transition phase diagram (below figure) The transition diagram, which is an FSM, starts with the dead state. In this state, there is no active carrier (at the physical layer) and the line is quiet. When one of the two nodes starts the communication, the connection goes into the establish state. In this state, options are negotiated between the two parties. If the two parties agree that they need authentication (for example, if they do not know each other), then the system needs to do authentication.

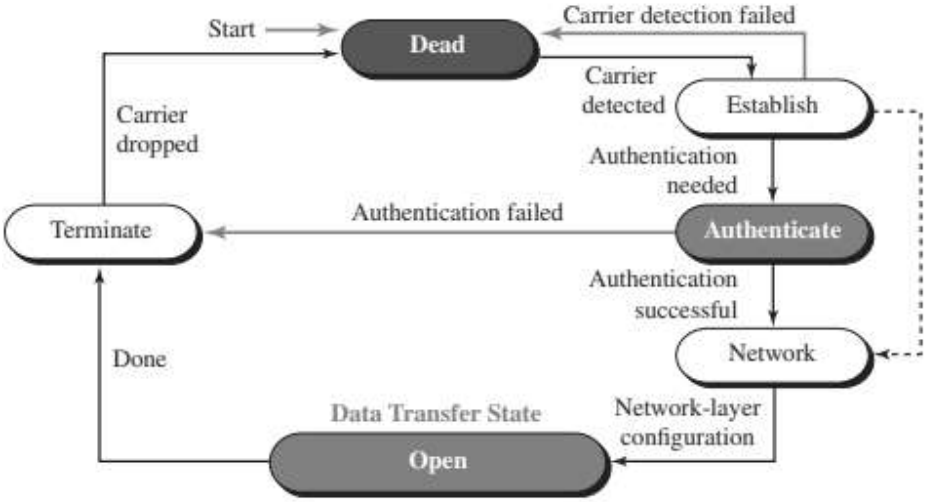

Fig. 2.11.2: Transition Phases

2.11.4 Multiplexing

Although PPP is a link-layer protocol, it uses another set of protocols to establish the link, authenticate the parties involved, and carry the network-layer data. Three sets of protocols are defined to make PPP powerful: the Link Control Protocol (LCP), two Authentication Protocols (APs), and several Network Control Protocols (NCPs). At any moment, a PPP packet can carry data from one of these protocols in its data field, as shown in below figure 2.11.3.

Link Control Protocol

The Link Control Protocol (LCP) is responsible for establishing, maintaining, configuring, and terminating links. It also provides negotiation mechanisms to set options between the two endpoints. Both endpoints of the link must reach an agreement about the options before the link can be established. See Figure 2.11.2.

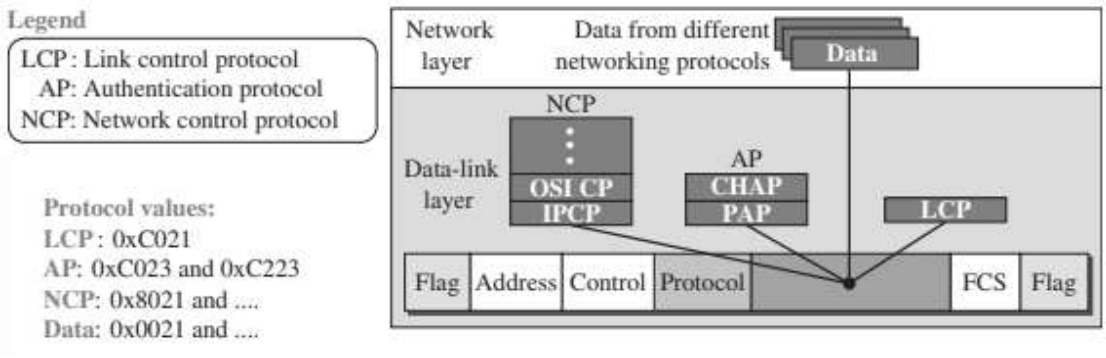

Fig. 2.11.3: Multiplexing in PPP

All LCP packets are carried in the payload field of the PPP frame with the protocol field set to C021 in hexadecimal (see Figure 2.11.4)

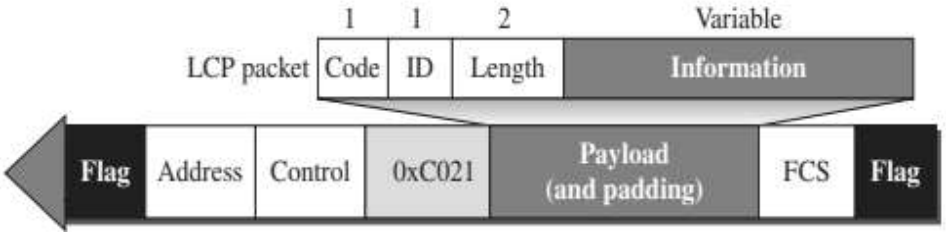

Fig. 2.11.4: LCP packet encapsulated in a frame

Code	Packet Type	Description
0x01	Configure-request	Contains the list of proposed options and their values
0x02	Configure-ack	Accepts all options proposed
0x03	Configure-nak	Announces that some options are not acceptable
0x04	Configure-reject	Announces that some options are not recognized
0x05	Terminate-request	Request to shut down the line
0x06	Terminate-ack	Accept the shutdown request
0x07	Code-reject	Announces an unknown code
0x08	Protocol-reject	Announces an unknown protocol
0x09	Echo-request	A type of hello message to check if the other end is alive
0x0A	Echo-reply	The response to the echo-request message
0x0B	Discard-request	A request to discard the packet

Table: LCP Packets

Authentication Protocols

Authentication plays a very important role in PPP because PPP is designed for use over dial-up links where verification of user identity is necessary. Authentication means validating the identity of a user who needs to access a set of resources. PPP has created two protocols for authentication: Password Authentication Protocol and Challenge Handshake Authentication Protocol. Note that these protocols are used during the authentication phase.

PAP

The **Password Authentication Protocol (PAP)** is a simple authentication procedure with a two-step process:

- The user who wants to access a system sends an authentication identification (usually the user name) and a password.
- The system checks the validity of the identification and password and either accepts or denies connection.

The three PAP packets are authenticate-request, authenticate-ack, and authenticate-nak. The first packet is used by the user to send the user name and password. The second is used by the system to allow access. The third is used by the system to deny access.

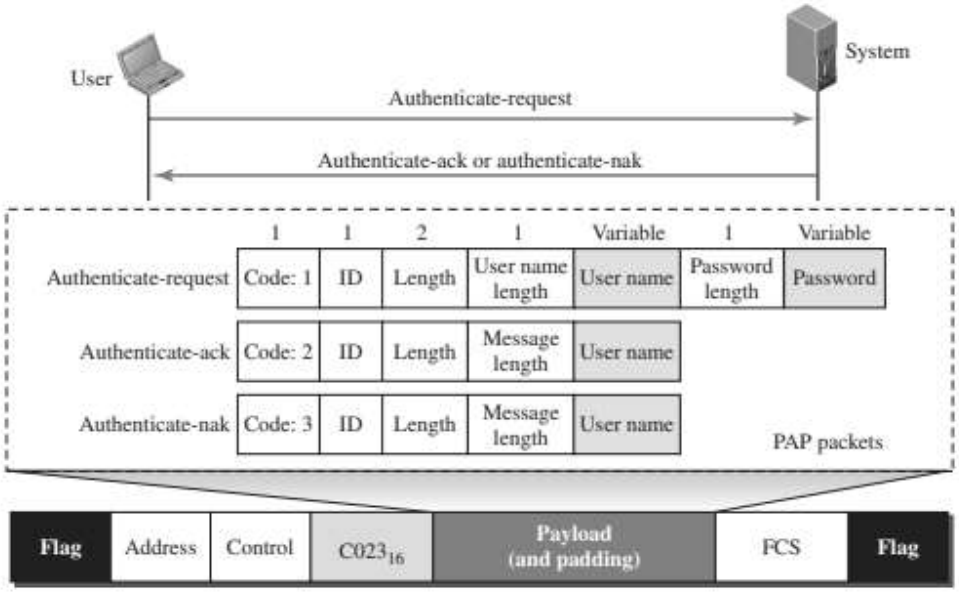

Fig. PAP Packets encapsulated in a PPP frame

CHAP

The **Challenge Handshake Authentication Protocol (CHAP)** is a three-way handshaking authentication protocol that provides greater security than PAP. In this method, the password is kept secret; it is never sent online.

- The system sends the user a challenge packet containing a challenge value, usually a few bytes.
- The user applies a predefined function that takes the challenge value and the user's own password and creates a result. The user sends the result in the response packet to the system.
- The system does the same. It applies the same function to the password of the user (known to the system) and the challenge value to create a result. If the result created is the same as the result sent in the response packet, access is granted.

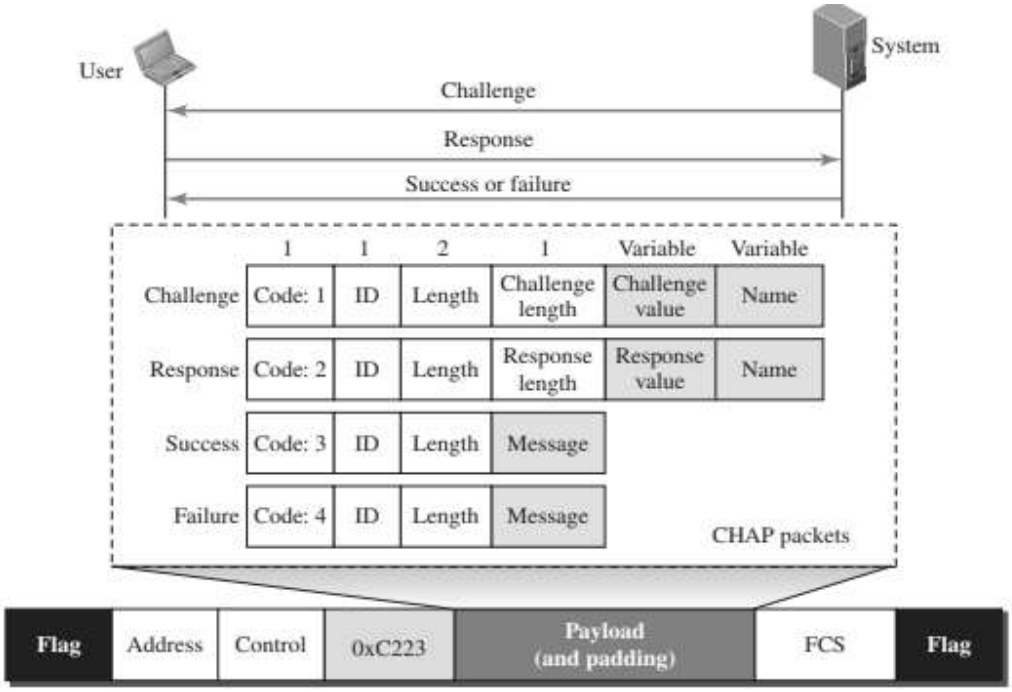

Fig. CHAP packets encapsulated in a PPP frame

Network Control Protocol

PPP is a multiple-network-layer protocol. It can carry a network-layer data packet from protocols defined by the Internet, OSI, Xerox, DECnet, AppleTalk, Novel, and so on. To do this, PPP has defined a specific Network Control Protocol for each network protocol. For example, IPCP (Internet Protocol Control Protocol) configures the link for carrying IP data packets. Xerox CP does the same for the Xerox protocol data packets, and so on. Note that none of the NCP packets carry network-layer data; they just configure the link at the network layer for the incoming data.

IPCP

One NCP protocol is the **Internet Protocol Control Protocol (IPCP).** This protocol configures the link used to carry IP packets in the Internet. IPCP is especially of interest to us. The format of an IPCP packet is shown in below Figure. Note that the value of the protocol field in hexadecimal is 8021.

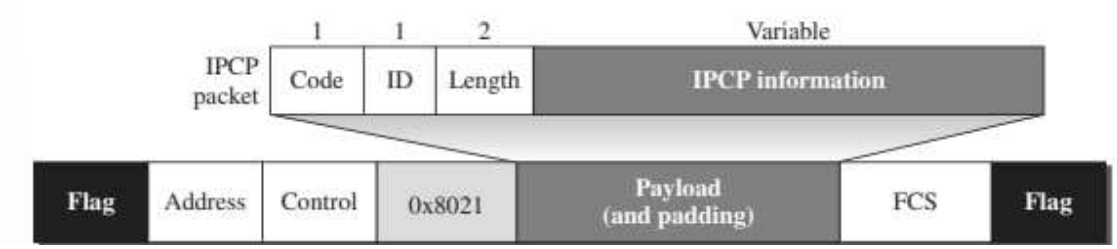

Fig. IPCP packet encapsulated in PPP frame

IPCP defines seven packets, distinguished by their code values, as shown in following Table.

Code	*IPCP Packet*
0x01	Configure-request
0x02	Configure-ack
0x03	Configure-nak
0x04	Configure-reject
0x05	Terminate-request
0x06	Terminate-ack
0x07	Code-reject

Other Protocols

There are other NCP protocols for other network-layer protocols. The OSI Network Layer Control Protocol has a protocol field value of 8023; the Xerox NS IDP Control Protocol has a protocol field value of 8025; and so on.

Data from the Network Layer

After the network-layer configuration is completed by one of the NCP protocols, the users can exchange data packets from the network layer. Here again, there are different protocol fields for different network layers.

Fig. IP Datagram encapsulated in a PPP frame

Multilink PPP

PPP was originally designed for a single-channel point-to-point physical link. The availability of multiple channels in a single point-to-point link motivated the development of Multilink PPP. In this case, a logical PPP frame is divided into several actual PPP frames. A segment of the logical frame is carried in the payload of an actual PPP frame, as shown in below figure. To show that the actual PPP frame is carrying a fragment of a logical PPP frame, the protocol field is set to (003d)16. This new development adds complexity. For example, a sequence number needs to be added to the actual PPP frame to show a fragment's position in the logical frame.

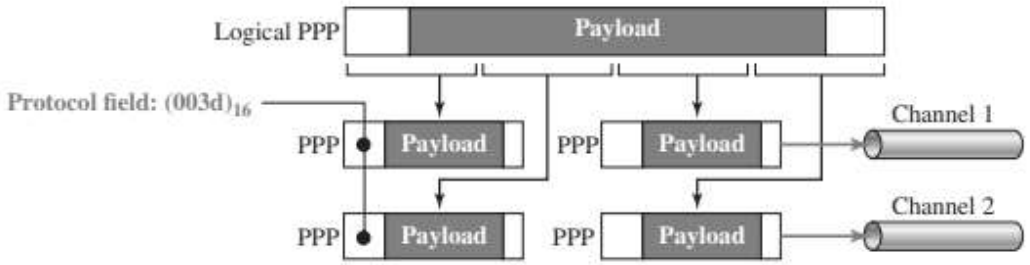

Fig. Multilink PPP

MEDIA ACCESS CONTROL

2.12 Random Access

The random-access methods we study in this chapter have evolved from a very interesting protocol known as **ALOHA**, which used a very simple procedure called multiple access (MA). The method was improved with the addition of a procedure that forces the station to sense the medium before transmitting. This was called carrier sense multiple access (CSMA). This method later evolved into two parallel methods: carrier sense multiple access with collision detection (CSMA/CD), which tells the station what to do when a collision is detected, and carrier sense multiple access with collision avoidance (CSMA/CA), which tries to avoid the collision.

2.12.1 Aloha

ALOHA, the earliest random access method, was developed at the University of Hawaii in early 1970. It was designed for a radio (wireless) LAN, but it can be used on any shared medium.

It is obvious that there are potential collisions in this arrangement. The medium is shared between the stations. When a station sends data, another station may attempt to do so at the same time. The data from the two stations collide and become garbled.

Pure ALOHA

The original ALOHA protocol is called pure ALOHA. This is a simple but elegant protocol. The idea is that each station sends a frame whenever it has a frame to send (multiple access). However, since there is only one channel to share, there is the possibility of collision between frames from different stations. Below Figure shows an example of frame collisions in pure ALOHA.

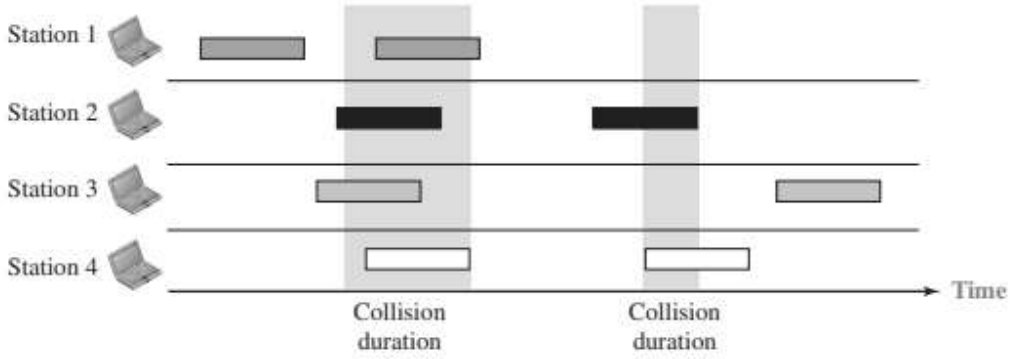

Fig. 2.12: Frames in a pure aloha network

There are four stations (unrealistic assumption) that contend with one another for access to the shared channel. The figure shows that each station sends two frames; there are a total of eight frames on the shared medium. Some of these frames collide because multiple frames are in contention for the shared channel. Figure shows that only two frames survive: one frame from station 1 and one frame from station 3. We need to mention that even if one bit of a frame coexists on the channel with one bit from another frame, there is a collision and both will be destroyed. It is obvious that we need to resend the frames that have been destroyed during transmission.

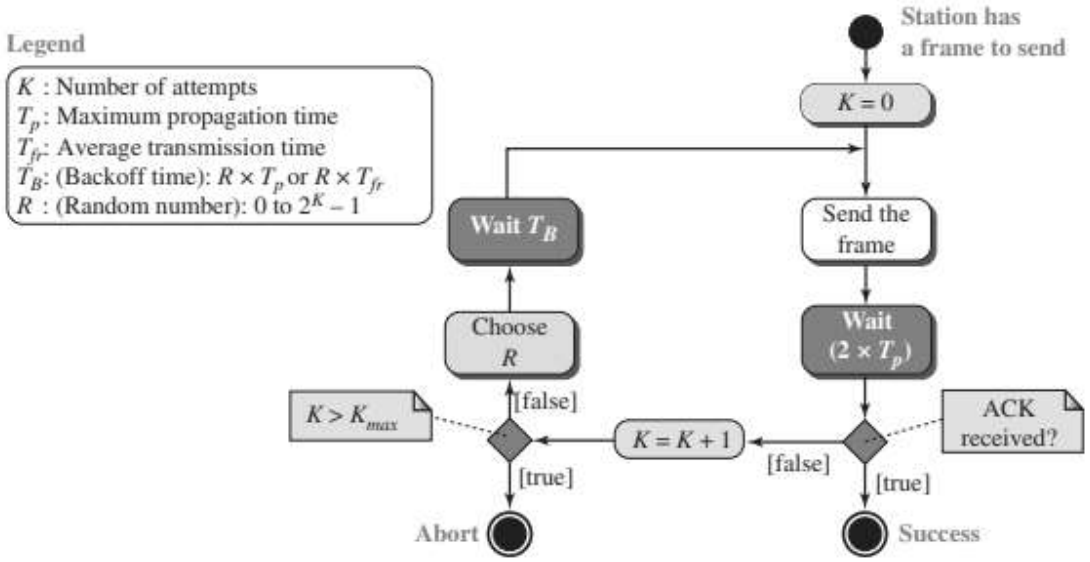

Fig. 2.12.1: procedure for pure aloha protocol

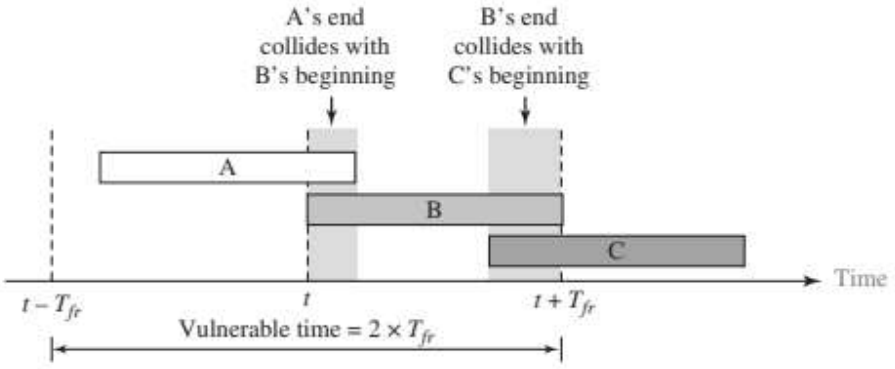

Fig. 2.12.2: Vulnerable time for pure aloha protocol

Pure ALOHA vulnerable time = $2 \times T_{fr}$

The throughput for pure ALOHA is $S = G \times e^{-2G}$.

The maximum throughput $S_{max} = 1/(2e) = 0.184$ when $G = (1/2)$.

Slotted ALOHA

Pure ALOHA has a vulnerable time of $2 \times T_{fr}$. This is so because there is no rule that defines when the station can send. A station may send soon after another station has started or just before another station has finished. Slotted ALOHA was invented to improve the efficiency of pure ALOHA.

In slotted ALOHA we divide the time into slots of T_{fr} seconds and force the station to send only at the beginning of the time slot. Below Figure shows an example of frame collisions in slotted ALOHA.

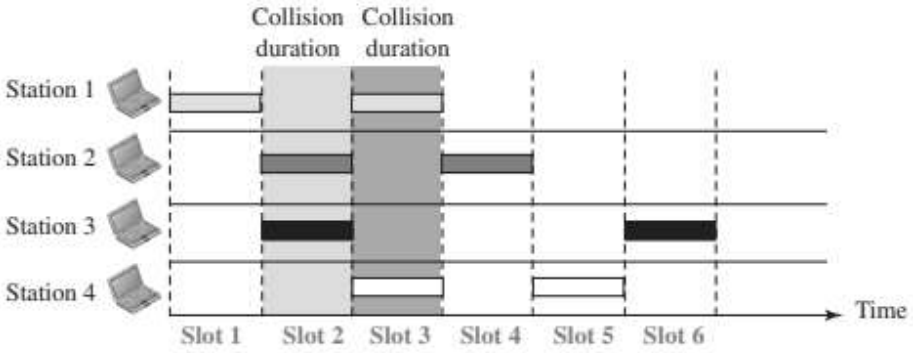

Fig. 2.12.3: Frames in a slotted aloha network

Because a station is allowed to send only at the beginning of the synchronized time slot, if a station misses this moment, it must wait until the beginning of the next time slot. This means that the station which started at the beginning of this slot has already finished sending its frame. Of course, there is still the possibility of collision if two stations try to send at the beginning of the same time slot. However, the vulnerable time is now reduced to one-half, equal to T_{fr}.

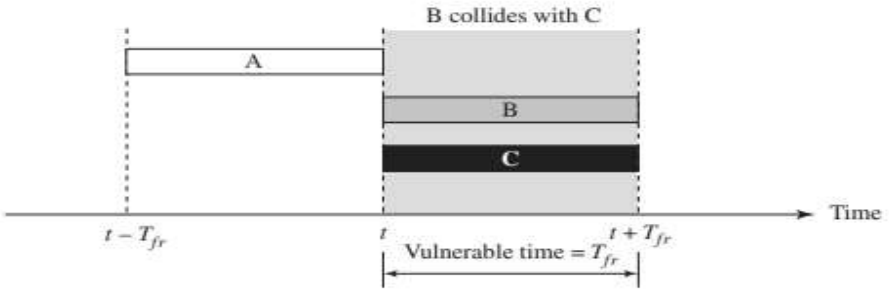

Fig. 2.12.4: Vulnerable time for slotted aloha protocol

Slotted ALOHA vulnerable time = T_{fr}

The throughput for slotted ALOHA is $S = G \times e^{-G}$.

The maximum throughput $S_{max} = 0.368$ when $G = 1$.

2.12.2 CSMA

To minimize the chance of collision and, therefore, increase the performance, the CSMA method was developed. The chance of collision can be reduced if a station senses the medium before trying to use it. **Carrier sense multiple access (CSMA)** requires that each station first listen to the medium (or check the state of the medium) before sending. In other words, CSMA is based on the principle "sense before transmit" or "listen before talk."

CSMA can reduce the possibility of collision, but it cannot eliminate it. The reason for this is shown in below Figure, a space and time model of a CSMA network. Stations are connected to a shared channel (usually a dedicated medium).

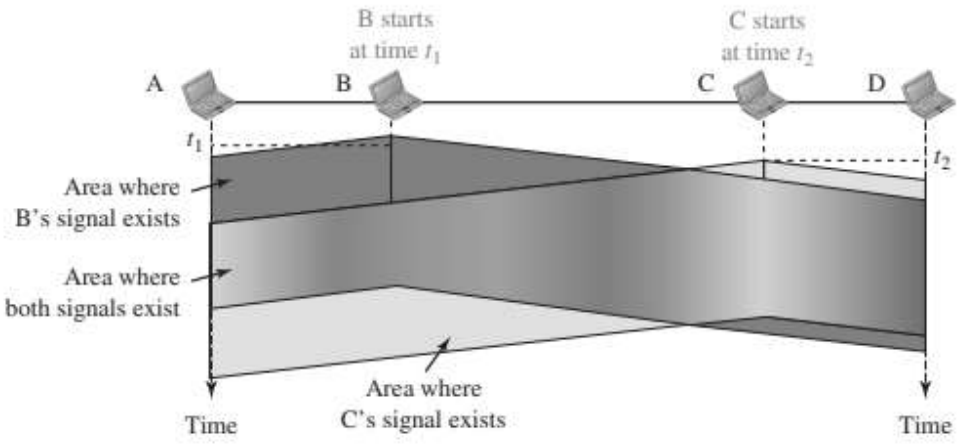

Fig. Space/time model of a collision in CSMA

At time t1, station B senses the medium and finds it idle, so it sends a frame. At time t2 (t2 > t1), station C senses the medium and finds it idle because, at this time, the first bits from station B have not reached station C. Station C also sends a frame. The two signals collide and both frames are destroyed.

Vulnerable Time

The vulnerable time for CSMA is the propagation time Tp. This is the time needed for a signal to propagate from one end of the medium to the other. When a station sends a frame and any other station tries to send a frame during this time, a collision will result. But if the first bit of the frame reaches the end of the medium, every station will already have heard the bit and will refrain from sending. Below Figure shows the worst case. The leftmost station, A, sends a frame at time t1, which reaches the rightmost station, D, at time t1 + Tp. The gray area shows the vulnerable area in time and space.

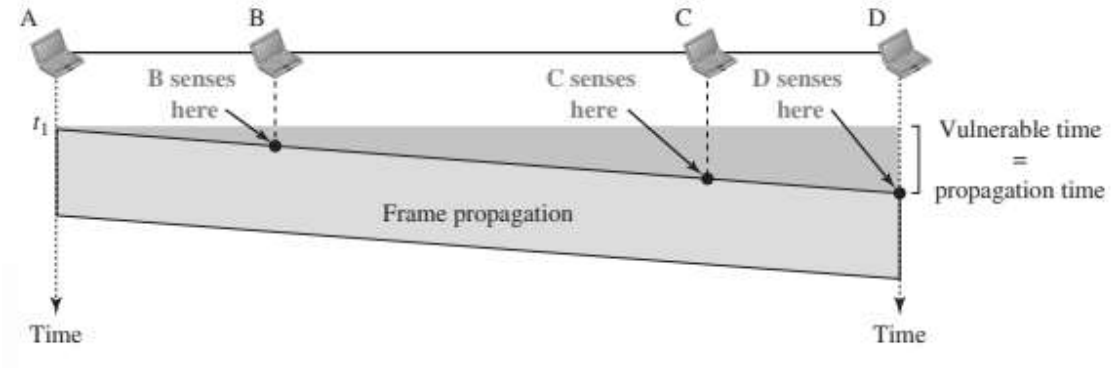

Fig. Vulnerable time in CSMA

Persistence Methods

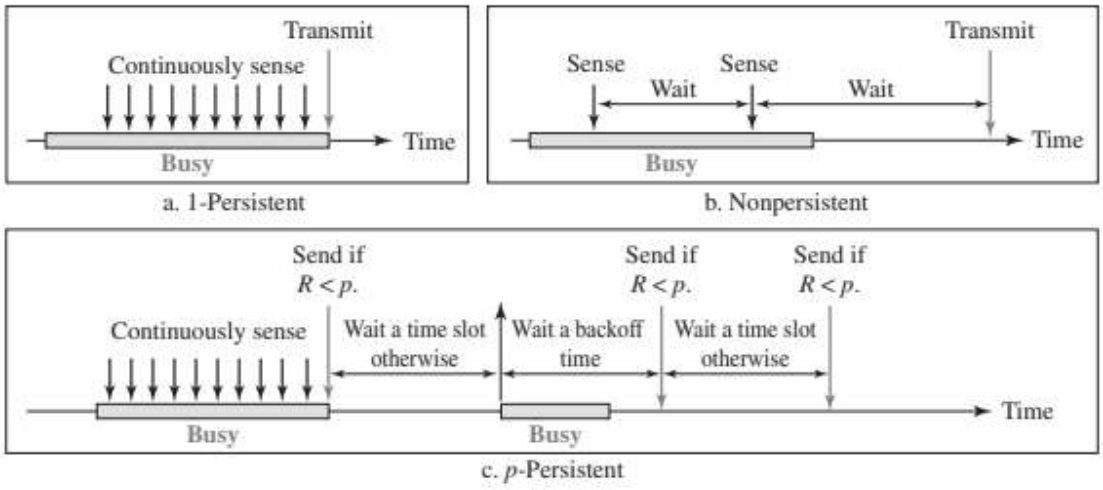

Fig. Behavior of three persistence methods

What should a station do if the channel is busy? What should a station do if the channel is idle? Three methods have been devised to answer these questions: the 1-persistent method, the nonpersistent method, and the p-persistent method. Figure below shows the behavior of three persistence methods when a station finds a channel busy.

Above figure shows the flow diagrams for these methods.

1-Persistent

The 1-persistent method is simple and straightforward. In this method, after the station finds the line idle, it sends its frame immediately (with probability 1). This method has the highest chance of collision because two or more stations may find the line idle and send their frames immediately. We will see later that Ethernet uses this method.

Nonpersistent

In the nonpersistent method, a station that has a frame to send senses the line. If the line is idle, it sends immediately. If the line is not idle, it waits a random amount of time and then senses the line again. The nonpersistent approach reduces the chance of collision because it is unlikely that two or more stations will wait the same amount of time and retry to send simultaneously. However, this method reduces the efficiency of the network because the medium remains idle when there may be stations with frames to send.

p-Persistent

The p-persistent method is used if the channel has time slots with a slot duration equal to or greater than the maximum propagation time. The p-persistent approach combines the advantages of the other two strategies. It reduces the chance of collision and improves efficiency. In this method, after the station finds the line idle it follows these steps:

1. With probability p, the station sends its frame.
2. With probability q = 1 − p, the station waits for the beginning of the next time slot and checks the line again.
    a. If the line is idle, it goes to step 1.

b. If the line is busy, it acts as though a collision has occurred and uses the backoff procedure.

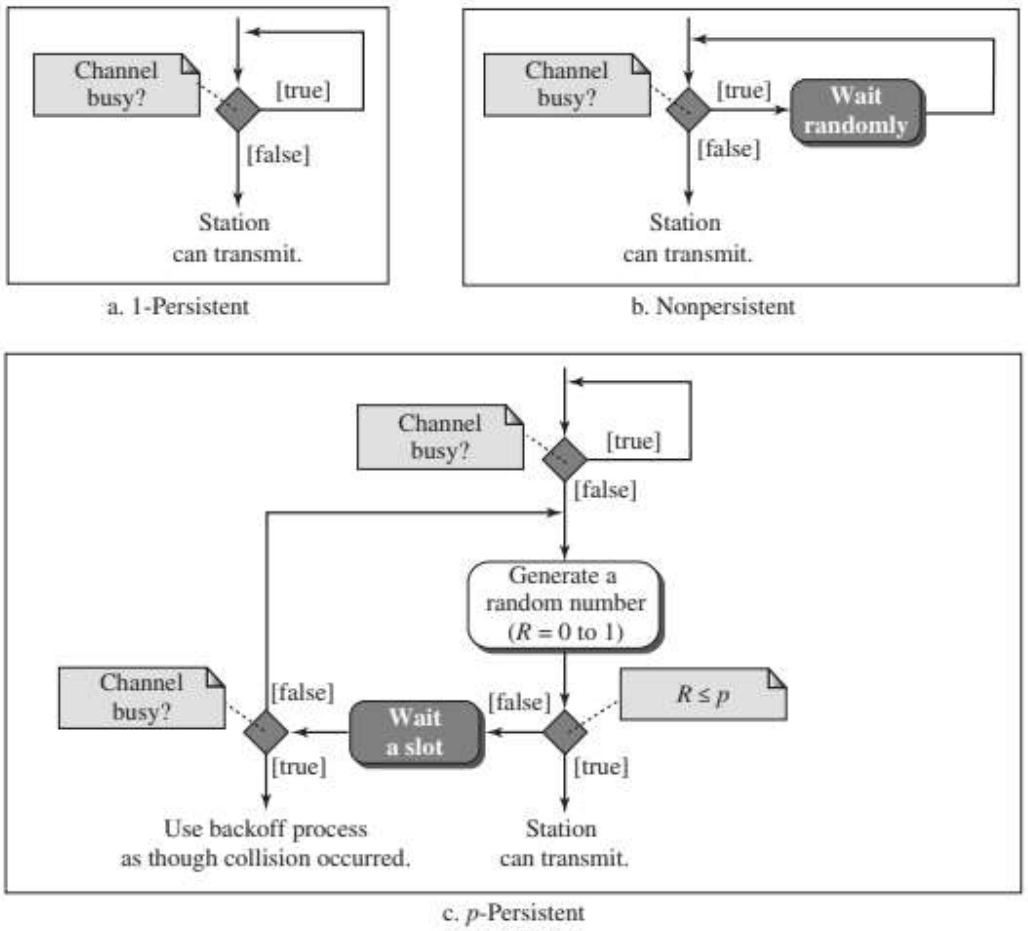

Fig. Flow diagram for three persistence methods

2.12.3 CSMA/CD

The CSMA method does not specify the procedure following a collision. Carrier sense multiple access with collision detection (CSMA/CD) augments the algorithm to handle the collision.

In this method, a station monitors the medium after it sends a frame to see if the transmission was successful. If so, the station is finished. If, however, there is a collision, the frame is sent again.

To better understand CSMA/CD, let us look at the first bits transmitted by the two stations involved in the collision. Although each station continues to send bits in the frame until it detects the

collision, we show what happens as the first bits collide. In below Figure, stations A and C are involved in the collision.

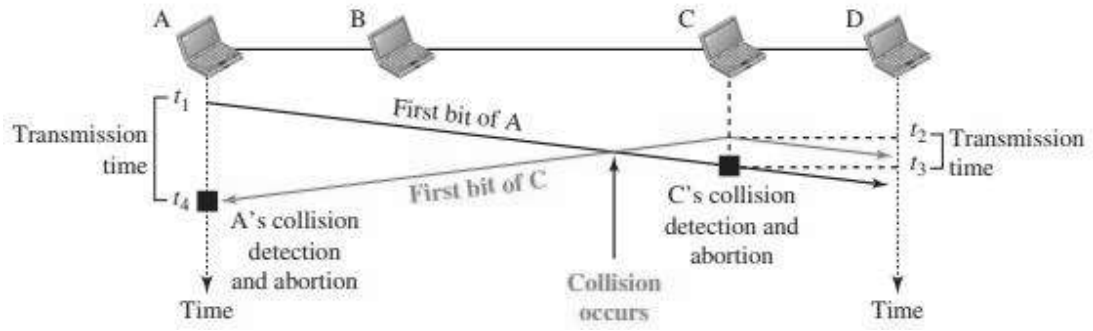

Fig. Collision of the first bits in CSMA/CD

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; it also immediately aborts transmission. Looking at the figure, we see that A transmits for the duration t4 − t1; C transmits for the duration t3 − t2.

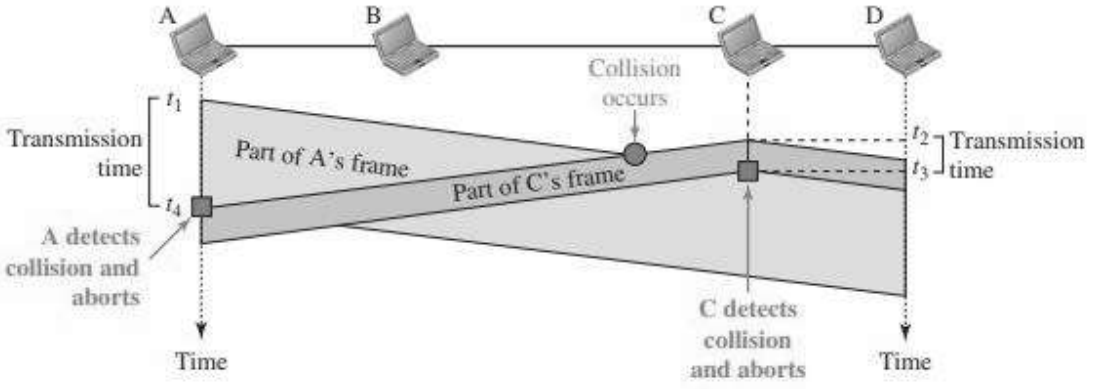

Fig. Collision and abortion in CSMA/CD

Minimum Frame Size

For CSMA/CD to work, we 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.

Procedure

Now let us look at the flow diagram for CSMA/CD in below Figure. It is similar to the one for the ALOHA protocol, but there are differences.

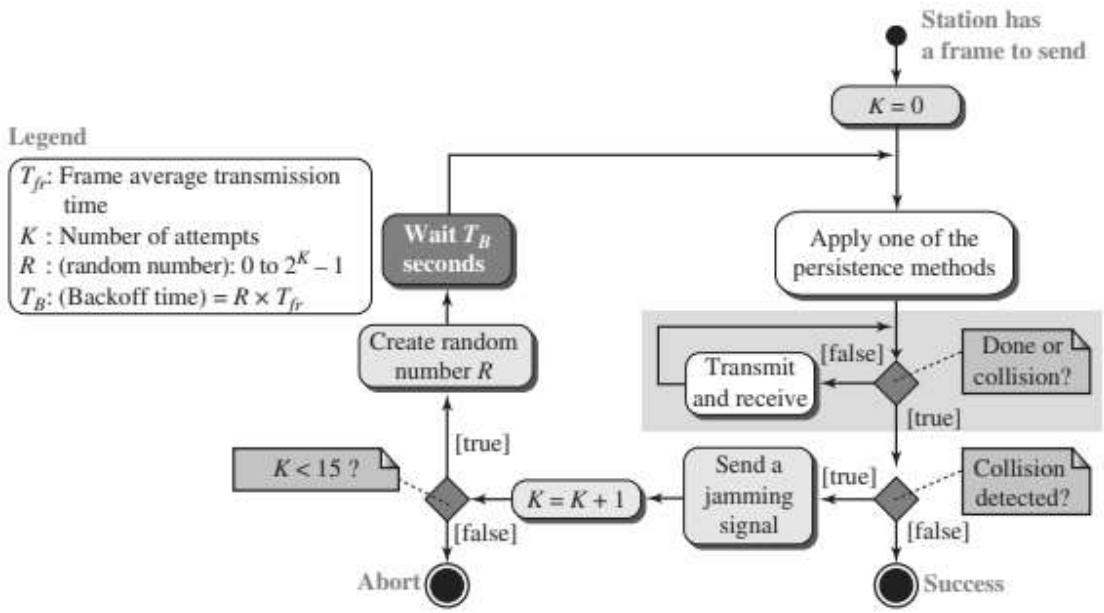

Fig. Flow diagram for the CSMA/CD

Energy Level

We can say that the level of energy in a channel can have three values: zero, normal, and abnormal. At the zero level, the channel is idle. At the normal level, a station has successfully captured the channel and is sending its frame. At the abnormal level, 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. Below Figure shows the situation.

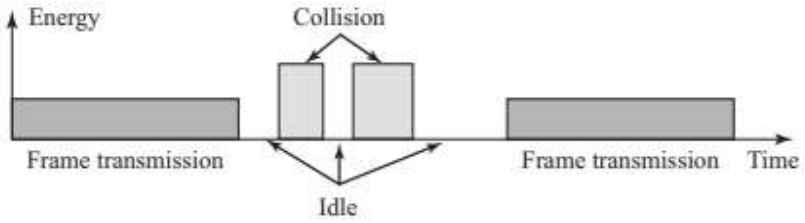

Fig. Energy level during transmission, idleness, or collision

Throughput

The throughput of CSMA/CD is 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

One of the LAN protocols that used CSMA/CD is the traditional Ethernet with the data rate of 10 Mbps. We discuss the Ethernet LANs in Chapter 13, but it is good to know that the traditional Ethernet was a broadcast LAN that used the 1-persistence method to control access to the common media. Later versions of Ethernet try to move from CSMA/CD.

2.12.4 CSMA/CA

Carrier sense multiple access with collision avoidance (CSMA/CA) was invented for wireless networks. Collisions are avoided through the use of CSMA/CA's three strategies: the interframe space, the contention window, and acknowledgments, as shown in below Figure 2.12.

- **Interframe Space (IFS).** First, collisions are avoided by deferring transmission even if the channel is found idle. When an idle channel is found, the station does not send immediately. It waits for a period of time called the interframe space or IFS.
- **Contention Window.** The contention window is an amount of time divided into slots. A station that is ready to send chooses a random number of slots as its wait time. As shown in figure 2.12.1.

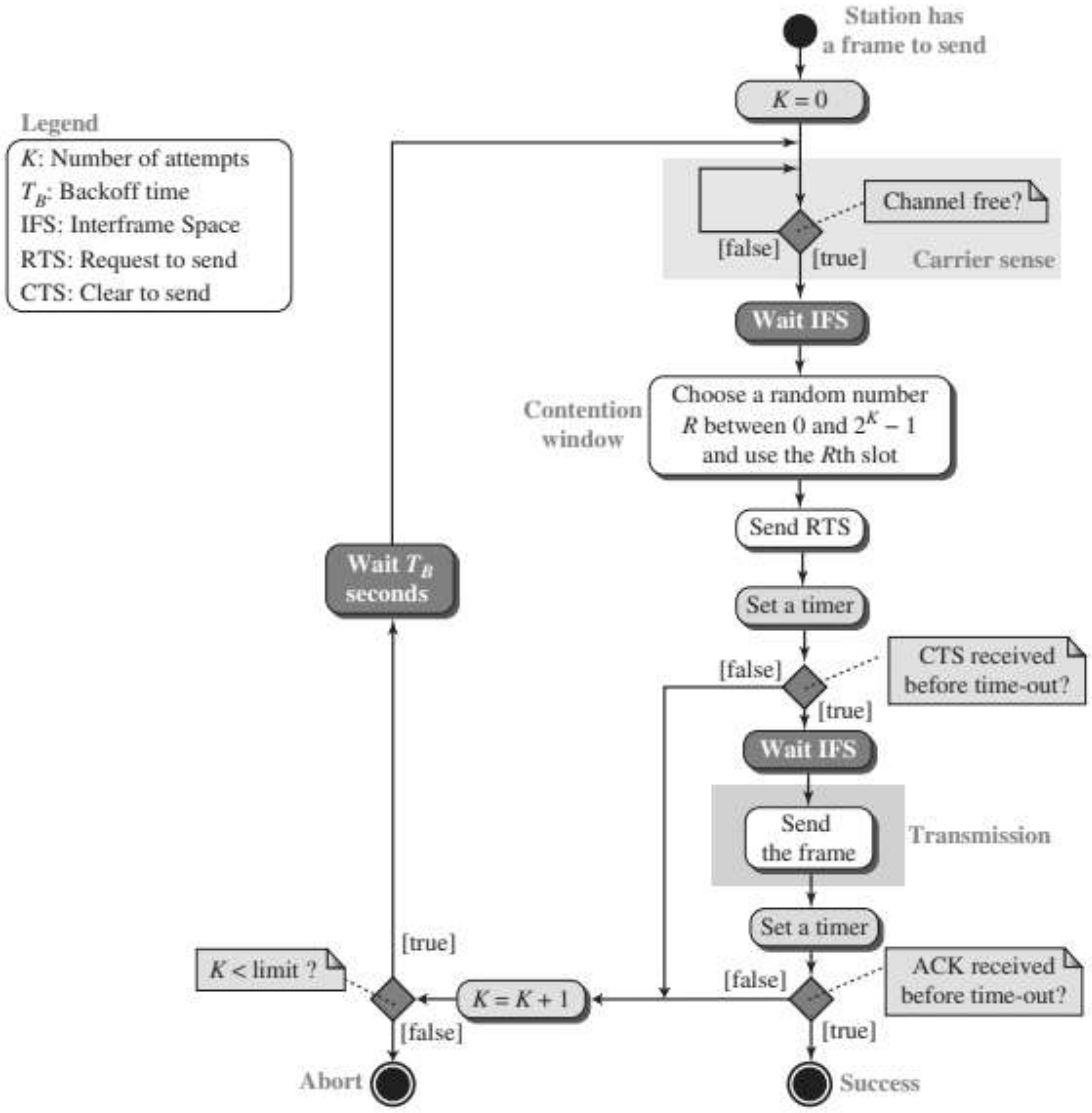

Fig. 2.12: Flow Diagram of CSMA/CA

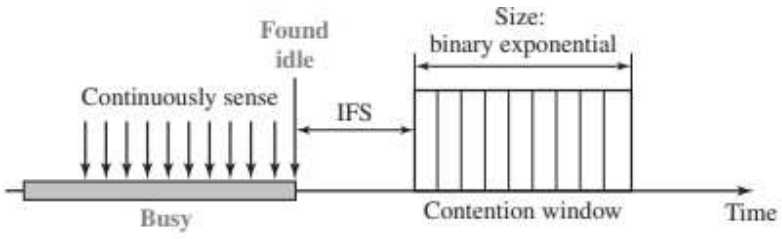

Fig. 2.12.1: Contention window

- **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.

Frame Exchange Time Line

Below Figure shows the 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.

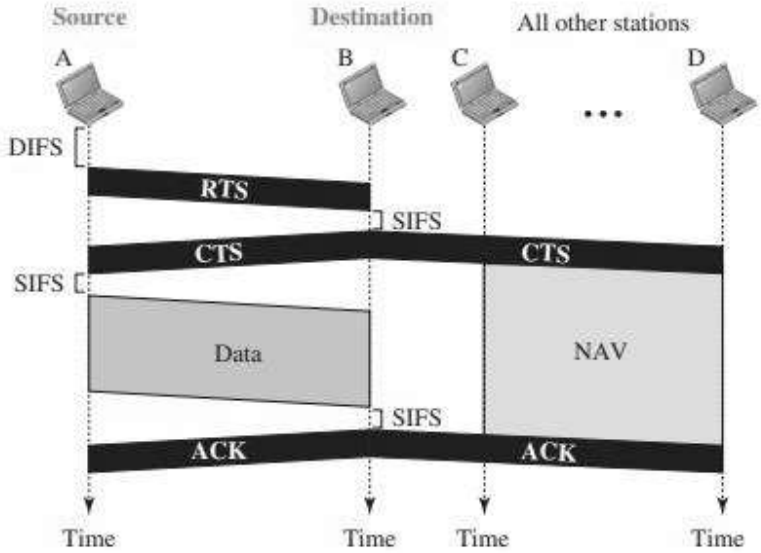

Fig. CSMA/CA and NAV

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 in CSMA/CD is a kind of indication to the source that data have arrived.

2.13 Controlled Access

In 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.

2.13.1 Reservation

In the reservation method, a station needs to make a reservation before sending data. Time is divided into intervals. In each interval, a reservation frame precedes the data frames sent in that interval.

If there are N stations in the system, there are exactly N reservation mini slots in the reservation frame. Each mini slot belongs to a station. When a station needs to send a data frame, it makes a reservation in its own mini slot.

Below Figure shows a situation with five stations and a five-mini slot 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.

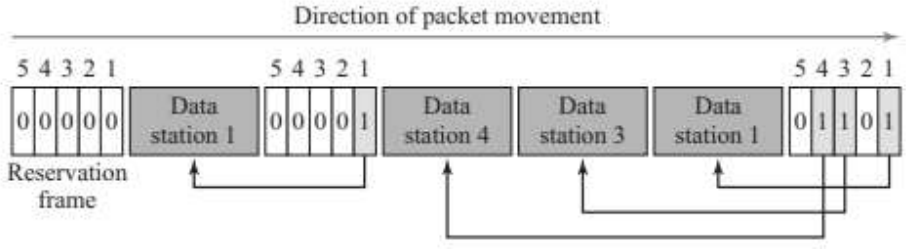

Fig. Reservations access method

2.13.2 Polling

Polling works with topologies in which one device is designated as a primary station and the other devices are secondary stations. All data exchanges must be made through the primary device even when the ultimate destination is a secondary device. The primary device controls the link; the secondary devices follow its instructions. It is up to the primary device to determine which device is allowed to use the channel at a given time. The primary device, therefore, is always the initiator of a session. This method uses poll and select functions to prevent collisions. However, the drawback is if the primary station fails, the system goes down.

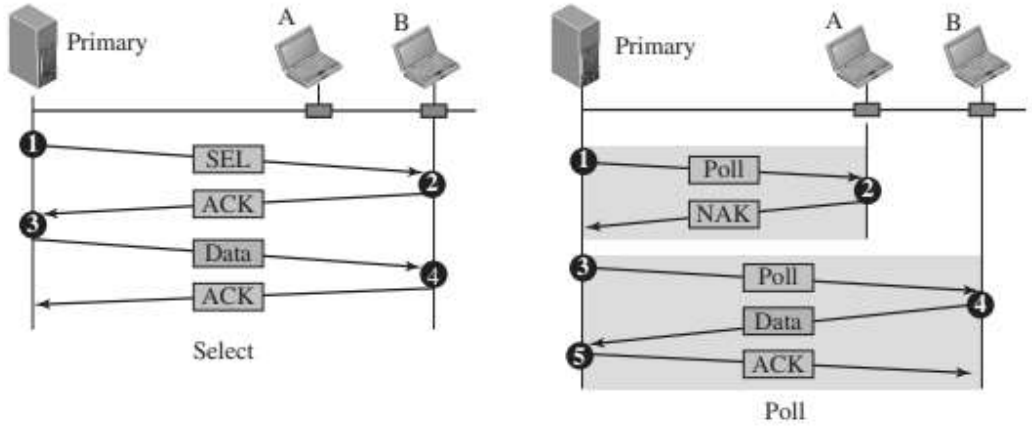

Fig. Select and poll functions in polling- access method

Select

The select function is used whenever the primary device has something to send. Remember that the primary controls the link. If the primary is neither sending nor receiving data, it knows the link is available. If it has something to send, the primary device sends it.

Poll

The poll function is 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.

2.13.3 Token Passing

In the token-passing method, the stations in a network are organized in a logical ring. In other words, for each station, there is 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.

Token management is needed for this access method. Stations must be limited in the time they can have possession of the token. 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. Another function of token management is to assign priorities to the stations and to the types of data being transmitted. And finally, token management is needed to make low-priority stations release the token to high-priority.

Logical Ring

In a token-passing network, stations do not have to be physically connected in a ring; the ring can be a logical one. Figure below shows four different physical topologies that can create a logical ring.

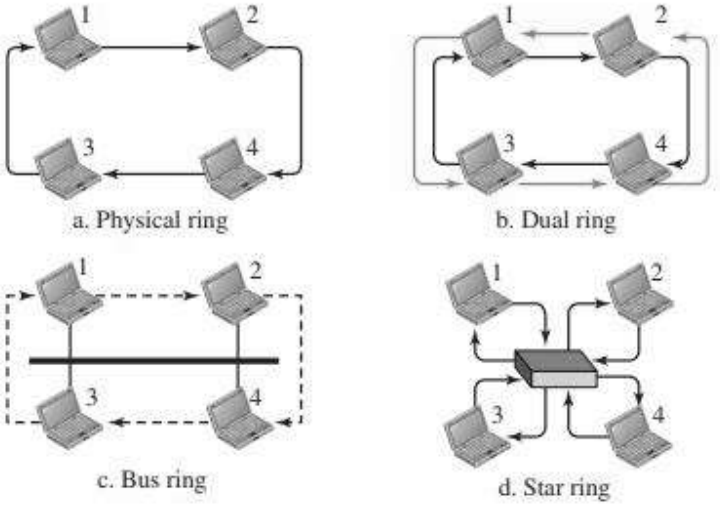

Fig. Logical ring and physical topology in token-passing access method