Unit –III

SYLLABUS: **Framing**: fixed size framing, variable size framing, , Flow control, Error control ,**Error detections Error correction**: block coding, linear block codes, **cyclic codes:** cyclic redundancy check, **, Checksum:** idea, one's complement internet check sum, services provided to Network Layer, **elementary Data link Layer protocols-** Unrestricted Simplex protocol, Simplex Stop-and-Wait Protocol, Simplex protocol for Noisy Channel.
Sliding window protocol: One bit, Go back N, Selective repeat-Stop and wait protocol, Data link layer in HDLC: configuration and transfer modes, frames, control field, point to point protocol (PPP): framing transition phase, multiplexing, multi link PPP.

INTRODUCTION:

- The data link layer needs to pack bits into frames, so that each frame is distinguishable from another.
- Our postal system practices a type of framing. The simple act of inserting a letter into an envelope separates one piece of information from another; the envelope serves as the delimiter.
- Framing in the data link layer separates a message from one source to a destination, or from other messages to other destinations, 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.

Fixed-Size Framing:

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 wide-area network, which uses frames of fixed size called cells.

Variable-Size Framing:

In variable-size framing, we need a way to define the end of the 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 Protocols:

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

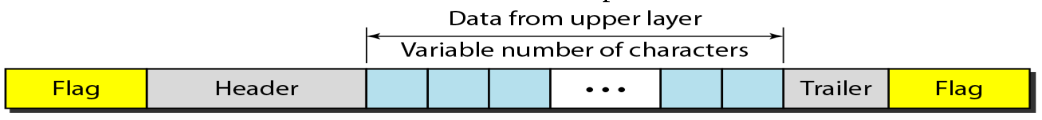
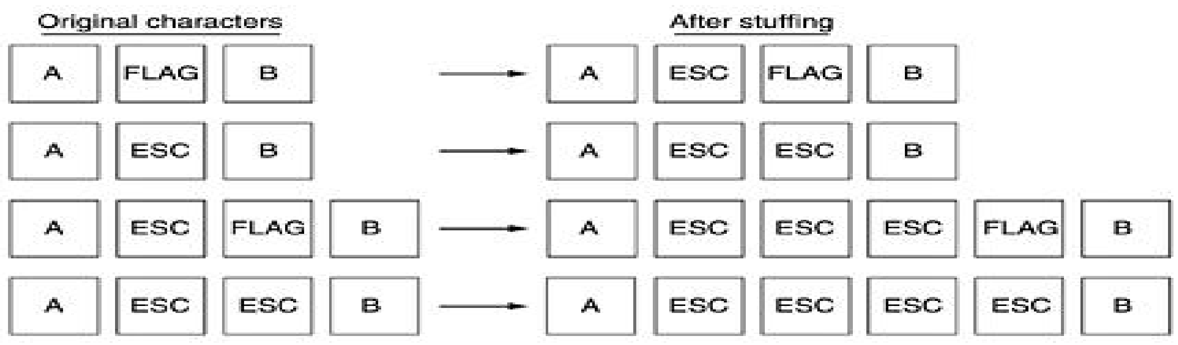


Figure: *A frame in a character-oriented protocol*

- Character-oriented framing was popular when only text was exchanged by the data link layers.
- The flag could be selected to be any character not used for text communication.
- Now, however, we send other types of information such as graphs, audio, and video. Any pattern used for the flag could also be part of the information. If this happens, the receiver, when it encounters this pattern in the middle of the data, thinks it has reached the end of the frame. To fix this problem, a byte-stuffing strategy was added to character-oriented framing.

Byte stuffing (or character stuffing):

- In this sender's data link layer insert a **special escape byte (ESC)** just before each **"accidental" flag** byte in the data.
- The data link layer on the receiving end removes the escape byte before the data are given to the network layer.
- This technique is called byte stuffing or character stuffing.
- Thus, a framing flag byte can be distinguished from one in the data by the absence or presence of an escape byte before it.
- Of course, the next question is: What happens if an escape byte occurs in the middle of the data?
- The answer is that it, too, is stuffed with an escape byte.
- Thus, any single escape byte is part of an escape sequence, whereas a doubled one indicates that a single escape occurred naturally in the data.



Major disadvantage of Character-oriented protocols is it use 8-bit characters. The universal coding systems in use today, such as Unicode, have 16-bit and 32-bit characters that conflict with 8-bit characters.

Bit-Oriented Protocols:

- ✓ In this data frames contains an arbitrary number of bits and allows character codes with an arbitrary number of bits per character.
- ✓ It works like this. Each frame begins and ends with a special bit pattern, **01111110** (in fact, a flag byte).
- ✓ Whenever the sender's data link layer encounters **five** consecutive **1**s in the data, it automatically stuffs a **0 bit into the outgoing bit stream**.
- ✓ This **bit stuffing** is analogous to byte stuffing, in which an escape byte is stuffed into the outgoing character stream before a flag byte in the data.
- ✓ When the receiver sees five consecutive incoming 1 bits, followed by a 0 bit, it automatically destuffs (i.e., deletes) the 0 bit.
- ✓ Just as byte stuffing is completely transparent to the network layer in both computers, so is bit stuffing.
- ✓ **If the user data contain the flag pattern, 01111110, this flag is transmitted as 011111010 but stored in the receiver's memory as 01111110**.

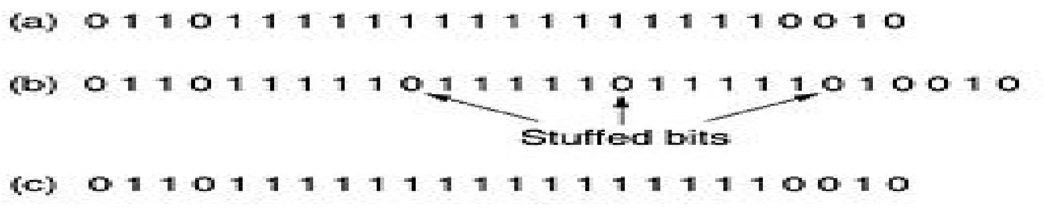

(a) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0
(b) 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0
Stuffed bits
(c) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

Fig: bit stuffing and unstuffing

FLOW AND ERROR CONTROL:

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

Flow Control:

- Flow control coordinates the amount of data that can be sent before receiving an acknowledgment and is one of the most important duties of the data link layer.
- In most protocols, flow control is a set of procedures that tells the sender how much data it can transmit before it must wait for an acknowledgment from the receiver.
- The flow of data must not be allowed to overwhelm the receiver.
- Any receiving device has a limited speed at which it can process incoming data and a limited amount of memory in which to store incoming data.
- The receiving device must be able to inform the sending device before those limits are reached and to request that the transmitting device send fewer frames or stop temporarily.
- Incoming data must be checked and processed before they can be used.
- The rate of such processing is often slower than the rate of transmission.
- For this reason, each receiving device has a block of memory, called a *buffer*, reserved for storing incoming data until they are processed.
- If the buffer begins to fill up, the receiver must be able to tell the sender to halt transmission until it is once again able to receive.

Error Control:

- Error control is both error detection and error correction.
- It allows the receiver to inform the sender of any frames lost or damaged in transmission and coordinates the retransmission of those frames by the sender.

- In the data link layer, the term *error control* refers primarily to methods of error detection and retransmission.
- Error control in the data link layer is often implemented simply: Any time an error is detected in an exchange, specified frames are retransmitted. This process is called **automatic repeat request (ARQ).**

ERROR DETECTIONS ERROR CORRECTION:

✓ Data can be corrupted during transmission. Some applications require that errors be detected and corrected.

Types of Errors:

Single-Bit Error:

The term *single-bit error* means that only 1 bit of a given data unit is changed from 1 to 0 or from 0 to 1.

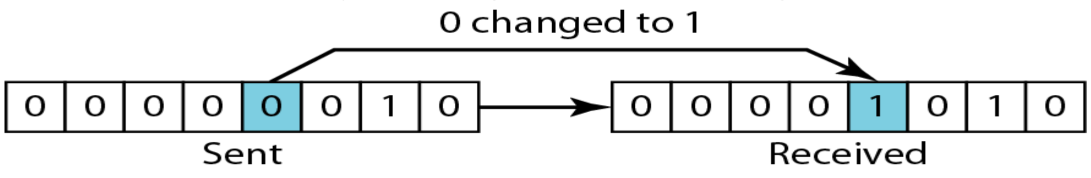


Figure: ***Single-bit error***

Burst Error

The term *burst error* means that 2 or more bits in the data unit have changed from 1 to 0 or from 0 to 1.

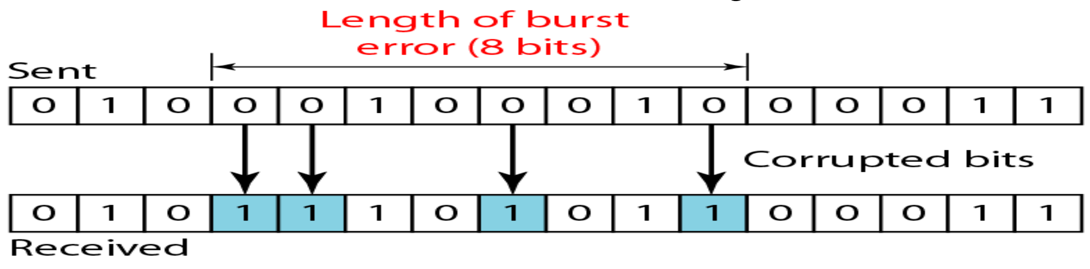


Figure: ***Burst error of length 8***

Redundancy:

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

Detection Versus Correction:

- The correction of errors is more difficult than detection.
- In error detection, we are looking only to see if any error has occurred. The answer is a simple yes or no.
- 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 message are important.

Forward Error Correction Versus Retransmission:

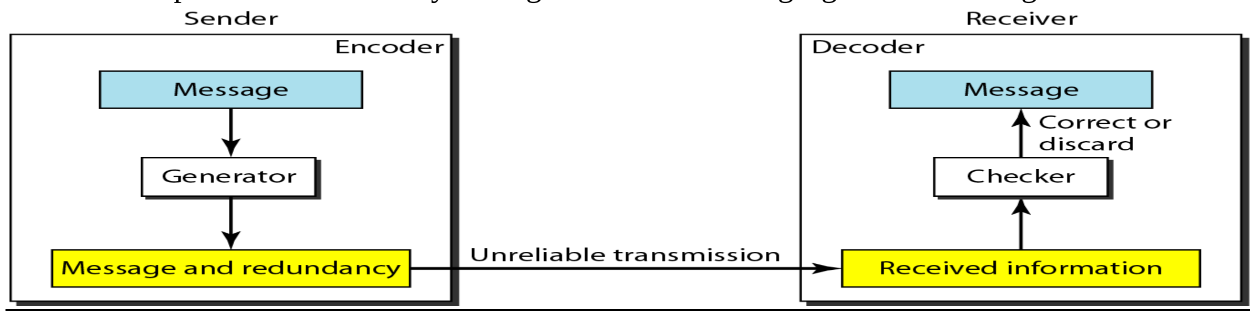
There are **two** main methods of **error correction**.

➢ **Forward error correction** is the process in which the receiver tries to guess the message by using redundant bits. This is possible if the number of errors is small.

➢ **Correction by retransmission** is a technique in which the receiver detects the occurrence of an error and asks the sender to resend the message. Resending is repeated until a message arrives that the receiver believes to be error-free.

Coding:

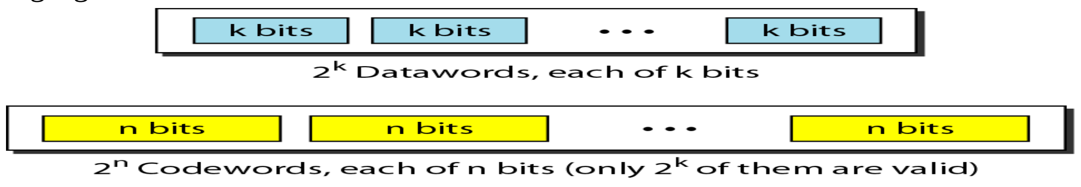
Redundancy is achieved through various coding schemes. The sender adds redundant bits through a process that creates a relationship between the redundant bits and the actual data bits. The receiver checks the relationships the two sets of bits to detect or correct the errors. The ratio of redundant bits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



Coding schemes can be divided into two broad categories: **Block Coding** and **Convolution Coding**.

<u>Block Coding:</u>

- In block coding, we divide our message into blocks, each of k bits, called **datawords**.
- We add r redundant bits to each block to make the total length to n = k + r. The resulting n-bit blocks are called **codewords**.
- With k bits, we can create a combination of 2^k datawords; with n bits, we can create a combination of 2^n codewords.
- The block coding process is a one-to-one; the same dataword is always encoded as the same codeword. This means that we have $2^n - 2^k$ codewords that are not used. We call this codewords invalid or illegal. Following figure shows the situation.



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

Figure:: datawords and codewords in block coding

Error Detection

If the following two conditions are met, the receiver can detect a change in the original codeword.
1. The receiver can find a list of valid codewords.
2. The original codeword has changed to an invalid one

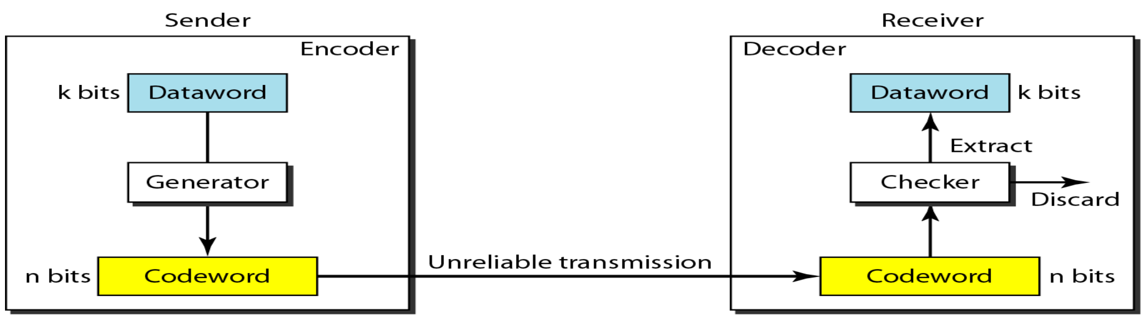


Figure: ***Process of error detection in block coding***

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

Error Correction

In error correction, the receiver needs to find (or guess) the original codeword sent. We can say that we need more redundant bits for error correction than for error detection.

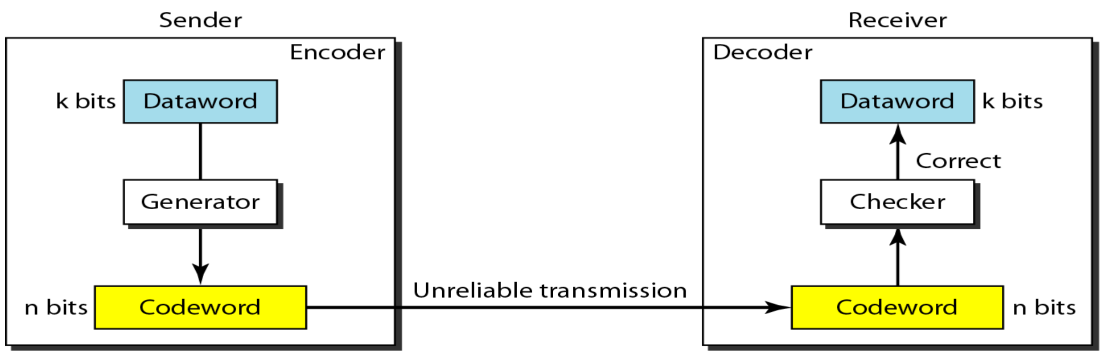


Figure 10.7 ***Structure of encoder and decoder in error correction***

Hamming Distance:

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

Example

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

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

Minimum Hamming Distance:

- Although the concept of the Hamming distance is the central point in dealing with error detection and correction codes, the measurement that is used **for designing a code** is the **minimum Hamming distance**.
- In a set of words, the minimum Hamming distance is the smallest Hamming distance between all possible pairs.
- We use $\mathbf{d_{min}}$ to define the minimum Hamming distance in a coding scheme.
- To find this value, we find the Hamming distances between all words and select the smallest one.

Example :

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

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

Table 1.1

Solution

We first find all Hamming distances.

d(000, 011) =2 d(000,101)=2 d(000,110)=2 d(011,101)=2 d(011,110)=2 d(101,110)=2

The d_{min} in this case is 2.

Example

Find the minimum Hamming distance of the coding scheme in Table

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

Table 1.2

Solution

We first find all the Hamming distances.

d(00000,01011)=3 d(00000,10101)=3 d(00000,11110)=4
d(01011,10101)=4 d(01011,11110)=3 d(10101,1110)=3

The d_{min} in this case is 3.

Three Parameters:

To mention any coding scheme we needs to have at least **three** parameters: the **codeword size n**, the **dataword size k**, and the **minimum Hamming distance d_{min}**. A coding scheme C is written as C(n, k) with a separate expression for $\mathbf{d_{min}}$. For example, we can call our first coding scheme C(3, 2) with $\mathbf{d_{min} = 2}$ and our second coding scheme C(5, 2) with $\mathbf{d_{min}}$ = 3.

Hamming Distance and Error

The relationship between the Hamming distance and errors occurring during transmission.

- When a **codeword is corrupted** during transmission, the Hamming distance between the sent and received codewords is the number of bits affected by the error.
- In other words, 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.

Minimum distance for Error Detection:

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

Example

The minimum Hamming distance for our first code scheme (Table 1.1) is 2. This code guarantees detection of only a single error. For example, if the third codeword (101) is sent and one error occurs, the received codeword does not match any valid codeword. If two errors occur, however, the received codeword may match a valid codeword and the errors are not detected.

Minimum Distance for Error Correction

To guarantee correction of up to **t** errors in all cases, the minimum Hamming distance in a block code must be $\mathbf{d_{min}} = 2t + 1$.

Example

A code scheme has a Hamming distance $\mathbf{d_{min}} = 4$. What is the error detection and correction capability of this scheme?

Solution

This code guarantees the detection of up to three errors (s = 3), but it can correct up to one error.In other words, if this code is used for error correction, part of its capability is wasted. Error correction codes need to have an odd minimum distance (3, 5, 7, . . .).

LINEAR BLOCK CODES:

- Almost all block codes used today belong to a subset called linear block codes.
- A linear block code is a code in which the exclusive OR (addition modulo-2) of two valid codewords creates another valid codeword.

Minimum Distance for Linear Block Codes:

- It is simple to find the minimum Hamming distance for a linear block code.
- The minimum Hamming distance is the number of 1s in the nonzero valid codeword with the smallest number of 1s.

Example

In our first code (Table 1), the numbers of 1s in the nonzero codewords are 2, 2, and 2. So the minimum Hamming distance is $\mathbf{d_{min}} = 2$. In our second code (Table 2), the numbers of 1s in the nonzero codewords are 3, 3, and 4. So in this code we have $\mathbf{d_{min}} = 3$.

Simple Parity- Check Code:

- In this code, a k-bit dataword is changed to an n-bit codeword where n = k + 1. The extra bit, called the parity bit, is selected to make the total number of 1s in the codeword even.
- The minimum Hamming distance for this category is $\mathbf{d_{min}} = 2$, which means that the code is a single-bit error-detecting code; it cannot correct any error.

Following table shows parity code with k=4 and n=5

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

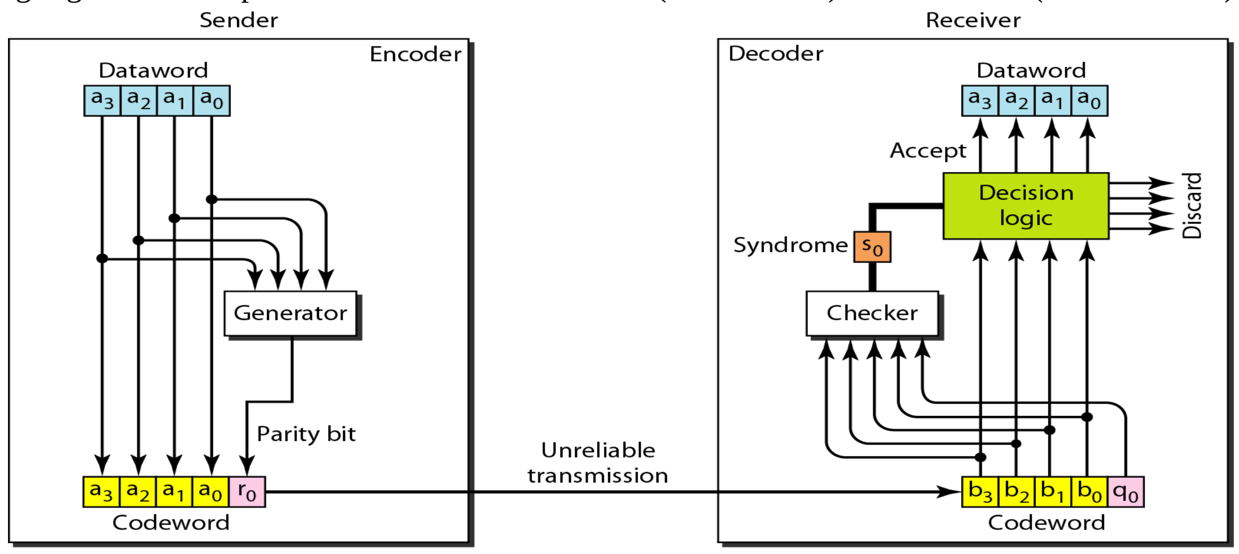


Figure: ***Encoder and decoder for simple parity-check code***

The encoder uses a generator that takes a copy of a 4-bit dataword (a_0,a_1,a_2, and a_3) and generates a parity bit r_0. The dataword bits and the parity bit create the 5-bit codeword. The parity bit that is added makes the number of 1s in the codeword even.

This is normally done by adding the 4 bits of the dataword (modulo-2); the result is the parity bit. In other words,

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

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.

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

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

Example

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

1. No error occurs; the received codeword is 10111. The syndrome is 0. The dataword 1011 is created.
2. One single-bit error changes a_1. The received codeword is 10011. The syndrome is 1. No dataword is created.
3. One single-bit error changes r_0.The received codeword is 10110. The syndrome is 1. No dataword is created. Note that although none of the dataword bits are corrupted, no dataword is created because the code is not sophisticated enough to show the position of the corrupted bit.
4. An error changes ro and a second error changes a_3. The received codeword is 00110. The syndrome is 0. The dataword 0011 is created at the receiver. Note that here the dataword is wrongly created due to the syndrome value. The simple parity-check decoder cannot detect an even number of errors. The errors cancel each other out and give the syndrome a value of 0.
5. Three bits- a_3, a_2, and a_1-are changed by errors. The received codeword is 01011. The syndrome is 1. The dataword is not created. This shows that the simple parity check, guaranteed to **detect one single error**, **can also find any odd number of errors.**

Two-dimensional parity check:

A better approach is the two-dimensional parity check. In this method, the dataword is organized in a table (rows and columns). Following Figure, the data to be sent, five 7-bit bytes, are put in separate rows. For each row and each column, 1 parity-check bit is calculated. The whole table is then sent to the receiver, which finds the syndrome for each row and each column. As Following Figure shows, the two-dimensional parity check can detect up to three errors that occur anywhere in the table (arrows point to the locations of the created nonzero syndromes). However, errors affecting 4 bits may not be detected.

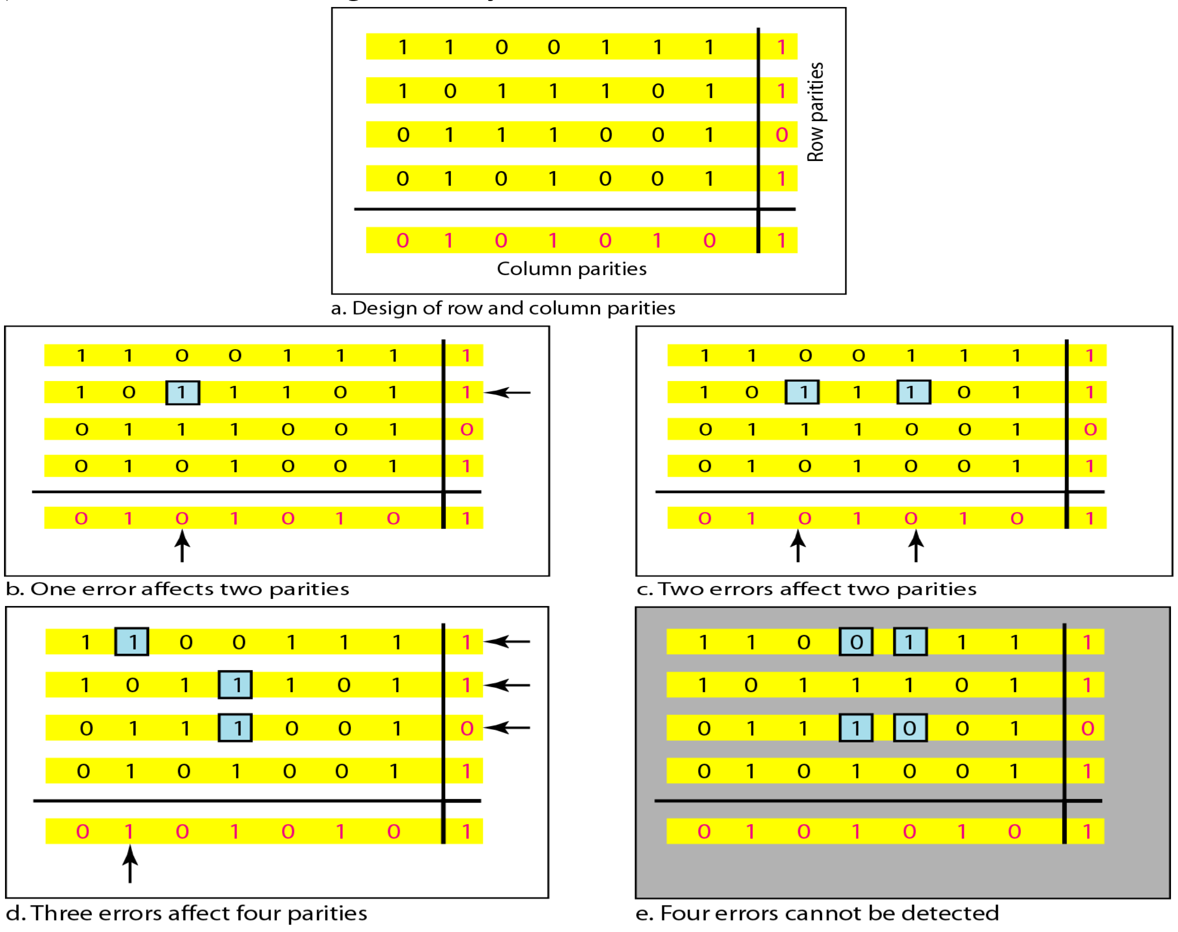

Row parities
Column parities
a. Design of row and column parities
b. One error affects two parities
c. Two errors affect two parities
d. Three errors affect four parities
e. Four errors cannot be detected

Figure: *two-dimensional parity-check code*

Hamming Codes

Now let us discuss a category of error-correcting codes called Hamming codes. These codes were originally designed with dmin = 3, which means that they can detect up to two errors or correct one single error. Although there are some Hamming codes that can correct more than one error, our discussion focuses on the single-bit error-correcting code. First let us find the relationship between n and k in a Hamming code. We need to choose an integer m >= 3. The values of nand k are then calculated from mas n = 2m – 1 and k = n - m. The number of check bits r =m.
For example, ifm =3, then n ::: 7 and k::: 4. This is a Hamming code C(7, 4) with dmin =3.
Figure shows the structure of the encoder and decoder for this example.

The structure ofthe encoder and decoder for a Hamming code
A copy of a 4-bit dataword is fed into the generator that creates three parity checks ro, r1 and r2 as shown below:

$$r0=a2+a1+a0$$
$$r1=a3+a2+a1$$
$$r2=a1+a0+a3$$

In other words, each of the parity-check bits handles 3 out of the 4 bits of the dataword. The total number of 1s in each 4-bit combination (3 dataword bits and 1 parity bit) must be even. We are not saying that these three equations are unique; any three equations that involve 3 of the 4 bits in the dataword and create independent equations (a combination of two cannot create the third) are valid.
The checker in the decoder creates a 3-bit syndrome (s2s1s0) in which each bit is the parity check for 4 out of the 7 bits in the received codeword:

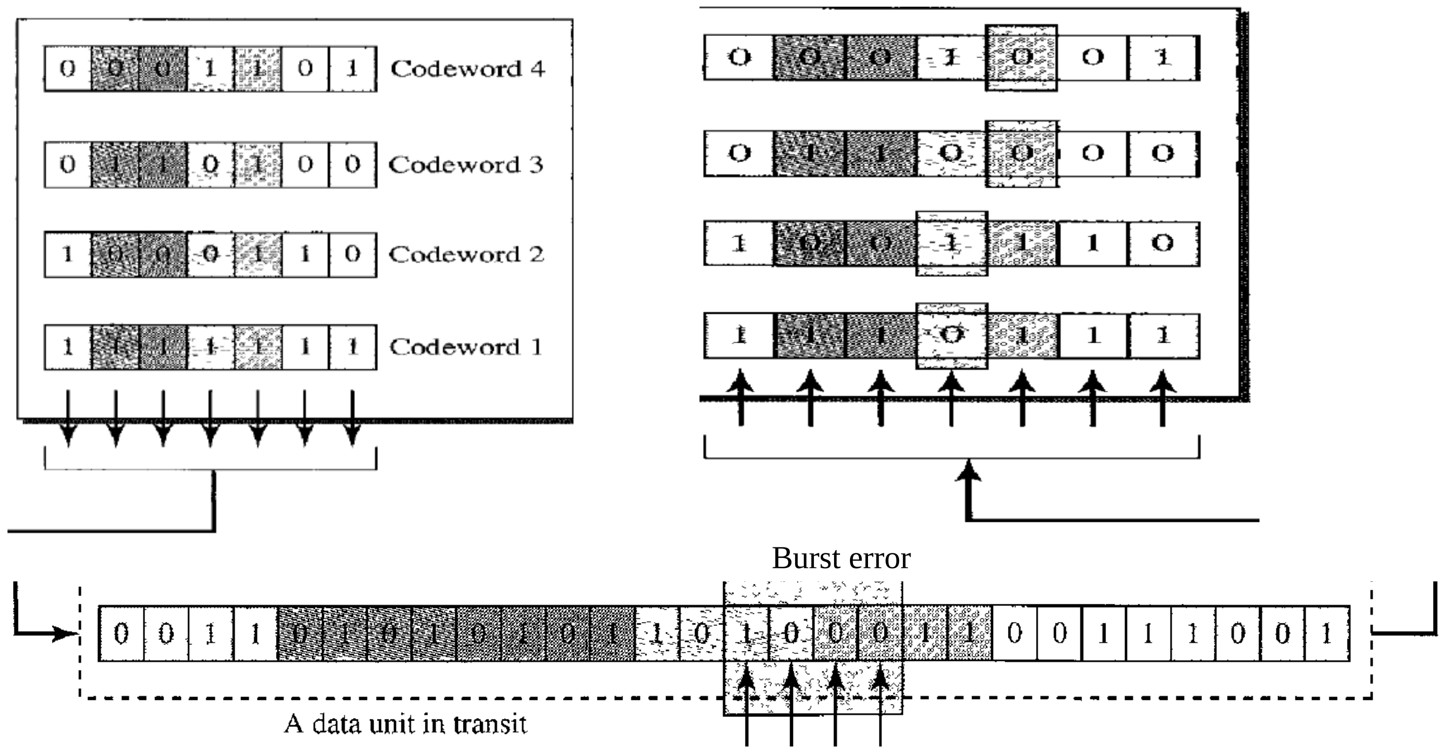
$$S0=b2+b1+b0+q0$$
$$S1=b3+b2+b1+q1$$
$$S3=b1+b0+b3+q2$$

The equations used by the checker are the same as those used by the generator with the parity-check bits added to the right-hand side of the equation. The 3-bit syndrome creates eight different bit patterns (000 to 111) that can represent eight different conditions.
These conditions define a lack of error or an error in 1 of the 7 bits of the received codeword, as shown in Table

Syndrome	000	001	010	011	100	101	110	111
Error	None	q0	ql	b2	q2	bo	b3	bl

Note that the generator is not concerned with the four cases shaded in Table because there is either no error or an error in the parity bit. In the other four cases, 1 of the bits must be flipped (changed from 0 to 1 or 1 to 0) to find the correct dataword.
The syndrome values in Table are based on the syndrome bit calculations. For example, if qo is in error, So is the only bit affected; the syndrome, therefore, is 001. If b2 is in error, So and s1 are the bits affected; the syndrome, therefore is OIl. Similarly, if bI is in error, all 3 syndrome bits are affected and the syndrome is 111.
Performance
A Hamming code can only correct a single error or detect a double error. However, there is a way to make it detect a burst error, as shown in Figure. The key is to split a burst error between several codewords, one error for each codeword. In data communications, we normally send a packet or a frame of data. To make the

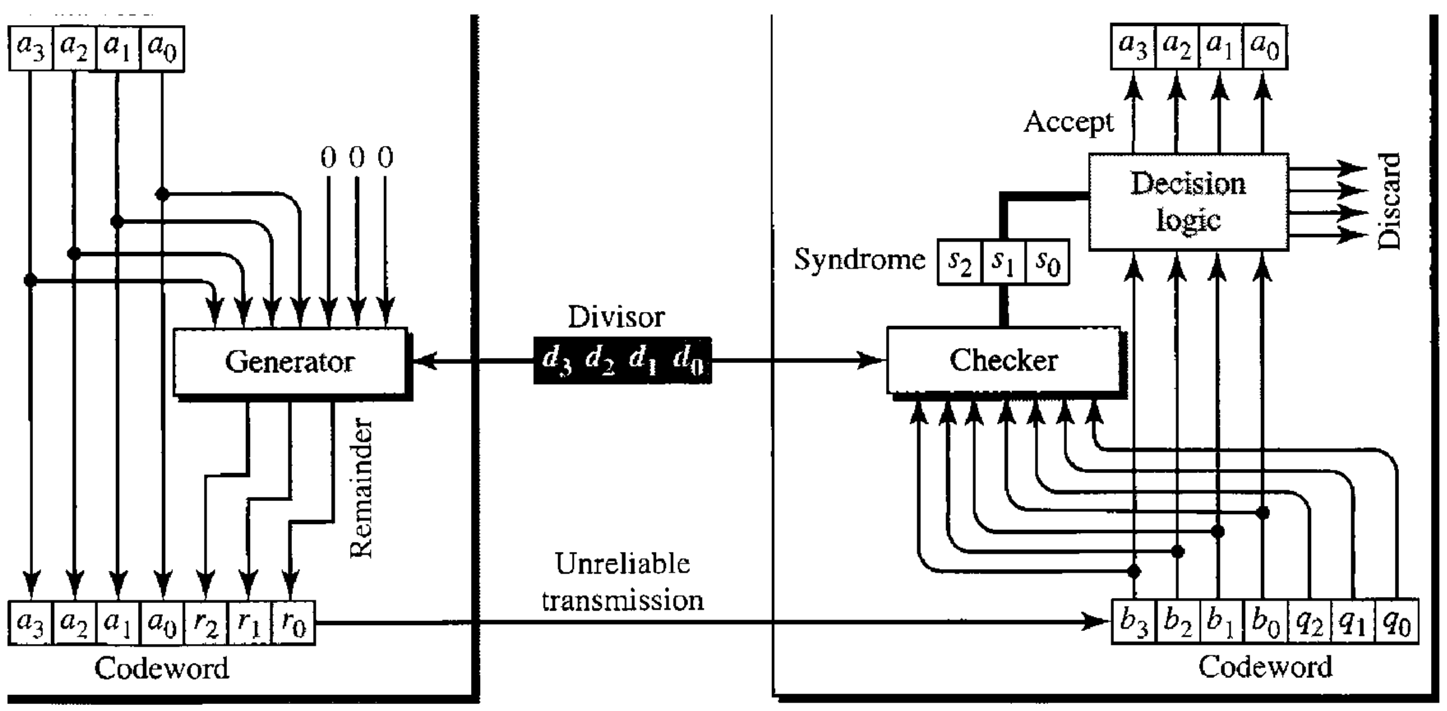
Hamming code respond to a burst error of size N, we need to make N codewords out of our frame. Then, instead of sending one codeword at a time, we arrange the codewords in a table and send the bits in the table a column at a time. In Figure 10.13, the bits are sent column by column (from the left). In each column, the bits are sent from the bottom to the top. In this way, a frame is made out of the four codewords and sent to the receiver.



Shows that when a burst error of size 4 corrupts the frame, only 1 bit from each codeword is corrupted. The corrupted bit in each codeword can then easily be corrected at the receiver.

Cyclic Redundancy Check

We can create cyclic codes to correct errors. Cyclic Redundancy Check (CRC) that is used in networks such as LANs and WANs. Figure shows CRC encoder and decoder



In the encoder, the dataword has k bits (4 here); the codeword has n bits (7 here). The size of the dataword is augmented by adding n - k (3 here) Os to the right-hand side of the word. The n-bit result is fed into the generator. The generator uses a divisor of size n - k + I (4 here), predefined and agreed upon. The generator divides the augmented dataword by the divisor (modulo-2 division). The quotient ofthe division is discarded; the remainder (r2rl ro) is appended to the dataword to create the codeword.

The decoder receives the possibly corrupted codeword. A copy of all n bits is fed to the checker which is a replica of the generator. The remainder produced by the checker is a syndrome of n - k (3 here) bits, which is fed to the decision logic analyzer. The analyzer has a simple function. If the syndrome bits are all as, the 4 leftmost bits of the codeword are accepted as the dataword (interpreted as no error); otherwise, the 4 bits
are discarded (error).
Encoder
Let us take a closer look at the encoder. The encoder takes the dataword and augments it with n - k number of as. It then divides the augmented dataword by the divisor, as shown in Figure .
Figure Division in CRC encoder

The process of modulo-2 binary division is the same as the familiar division process we use for decimal numbers. However, as mentioned at the beginning of the chapter, in this case addition and subtraction are the same. We use the XOR operation to do both.
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-Os divisor. When there are no bits left to pull down, we have a result. The 3-bit remainder forms the check bits (r2' rl' and ro). They are appended to the dataword to create the codeword.
Decoder
The codeword can change during transmission. The decoder does the same division process as the encoder. The remainder of the division is the syndrome. If the syndrome is all Os, there is no error; the dataword is separated from the received codeword and accepted. Otherwise, everything is discarded. Figure 10.16 shows two cases: The left hand figure shows the value of syndrome when no error has occurred; the syndrome is 000. The right-hand part of the figure shows the case in which there is one single error.The syndrome is not all Os (it is OIl).
Figure Division in the CRC decoder for two cases

CHECKSUM

The last error detection method we discuss here is called the checksum. The checksum is used in the Internet by several protocols although not at the data link layer.

Idea

The concept of the checksum is not difficult. Let us illustrate it with a few examples.

Example

Suppose our data is a list of five 4-bit numbers that we want to send to a destination. In addition to sending these numbers, we send the sum of the numbers. For example, if the set of numbers is (7, 11, 12, 0, 6), we send (7, 11, 12,0,6,36), where 36 is the sum of the original numbers. The receiver adds the five numbers and compares the result with the sum. If the two are the same, the receiver assumes no error, accepts the five numbers, and discards the sum. Otherwise, there is an error somewhere and the data are not accepted.

Example

We can make the job of the receiver easier if we send the negative (complement) of the sum, called the checksum. In this case, we send (7, 11, 12,0,6, -36). The receiver can add all the numbers received (including the checksum). If the result is 0, it assumes no error; otherwise, there is an error.

One's Complement

The previous example has one major drawback. All of our data can be written as a 4-bit word (they are less than 15) except for the checksum. One solution is to use one's complement arithmetic. In this arithmetic, we can represent unsigned numbers between 0 and 2n - 1 using only n bits. t If the number has more than n bits, the extra leftmost bits need to be added to the n rightmost bits (wrapping). In one's complement arithmetic, a negative number can be represented by inverting all bits (changing a 0 to a 1 and a 1 to a 0). This is the same as subtracting the number from 2n - 1.

Example

How can we represent the number 21 in one's complement arithmetic using only four bits?
t Although one's complement can represent both positive and negative numbers, we are concerned only with unsigned representation here.
Solution
The number 21 in binary is 10101 (it needs five bits). We can wrap the leftmost bit and add it to the four rightmost bits. We have (0101 + 1) = 0110 or 6.

Example

Let us redo Exercise 10.19 using one's complement arithmetic. The sender initializes the checksum to 0 and adds all data items and the checksum (the checksum is considered as one data item and is shown in color). The result is 36. However, 36 cannot be expressed in 4 bits. The extra two bits are wrapped and added with the sum to create the wrapped sum value 6. In the figure, we have shown the details in binary. The sum is then omplemented, resulting in the checksum value 9 (15 - 6 = 9). The sender now sends six data items to the receiver including the checksum 9. The receiver follows the same procedure as the sender. It adds all data items (including the checksum); the result is 45. The sum is wrapped and becomes 15. The wrapped sum is complemented and becomes O. Since the value of the checksum is 0, this means that the data is not corrupted. The receiver drops the checksum and keeps the other data items. If the checksum is not zero, the entire packet is dropped.

Internet Checksum

Traditionally, the Internet has been using a 16-bit checksum. The sender calculates the checksum by following these steps.

Sender site:

1. The message is divided into 16-bit words.
2. The value of the checksum word is set to O.
3. All words including the checksum are added ushtg one's complement addition.
4. The sum is complemented and becomes the checksum.
5. The checksum is sent with the data.

The receiver uses the following steps for error detection.

Receiver site:

1. The message (including checksum) is divided into 16-bit words.
2. All words are added using one's complement addition.
3. The sum is complemented and becomes the new checksum.
4. If the value of checksum is 0, the message is accepted; otherwise, it is rejected.

Example Let us calculate the checksum for a text of 8 characters ("Forouzan"). The text needs to be divided into 2-byte (l6-bit) words. We use ASCII (see Appendix A) to change each byte to a 2-digit hexadecimal number. For example, F is represented as Ox46 and 0 is represented as Ox6F. Figure shows how the checksum is calculated at the sender and receiver sites. In part a of the figure, the value of partial sum for the first column is Ox36. We keep the rightmost digit (6) and insert the Figure

I 0 1 3 Carries	1 () 1 3 Carries
4 6 6 F (Fo)	4 6 6 F IFo)
7 2 6 F (ro)	7 2 6 F (ro)
7 5 7 A luz)	7 5 7 A (uz)
6 1 6 E (an)	6 1 6 E (an)
0 0 0 0 Checksum (initial)	7 0 3 8 Checksum (received)
8 F C 6 Sum (partial)	F F F E Sum (partial)
1	1
8 F C 7 Sum	F F F F Sum
7 0 3 8 Checksum (to send)	0 0 0 () Checksum (new}
a. Checksum at the sender site	b. Checksum at the receiver site

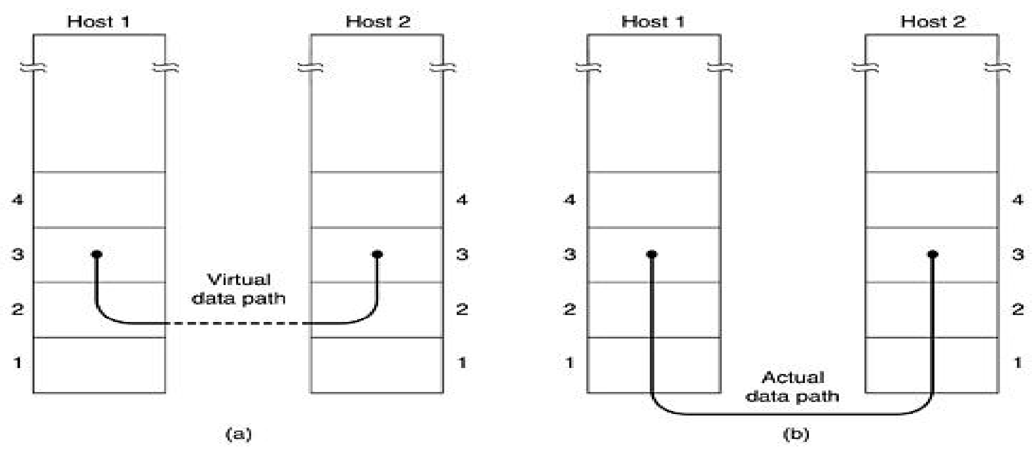
<u>Performance</u>

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.

<u>SERVICES PROVIDED TO NETWORK LAYER:</u>

The function of the data link layer is to provide services to the network layer. The principal service is transferring data from the network layer on the source machine to the network layer on the destination machine. On the source machine is an entity, call it a process, in the network layer that hands some bits to the data link layer for transmission to the destination. The job of the data link layer is to transmit the bits to the destination machine so they can be handed over to the network layer there, as shown in Fig. (a). The actual transmission follows the path of Fig. (b), but it is easier to think in terms of two data link layer processes communicating using a data link protocol.

ıre 3-2. (a) Virtual communication. (b) Actual communication.



The data link layer can be designed to offer various services. The actual services offered can vary from system to system. Three reasonable possibilities that are commonly provided are

1. Unacknowledged connectionless service.
2. Acknowledged connectionless service.
3. Acknowledged connection-oriented service.

<u>Unacknowledged connectionless service</u>:

- ➢ It consists of having the source machine send independent frames to the destination machine without having the destination machine acknowledge them.
- ➢ No logical connection is established beforehand or released afterward.
- ➢ If a frame is lost due to noise on the line, no attempt is made to detect the loss or recover from it in the data link layer.

- This class of service is appropriate when the error rate is very low so that recovery is left to higher layers.
- It is also appropriate for real-time traffic, such as voice, in which late data are worse than bad data.
- Most LANs use unacknowledged connectionless service in the data link layer.

Acknowledged Connectionless Service:

- When this service is offered, there are still no logical connections used, but each frame sent is individually acknowledged.
- In this way, the sender knows whether a frame has arrived correctly.
- If it has not arrived within a specified time interval, it can be sent again.
- This service is useful over unreliable channels, such as wireless systems.

Connection-Oriented Service:

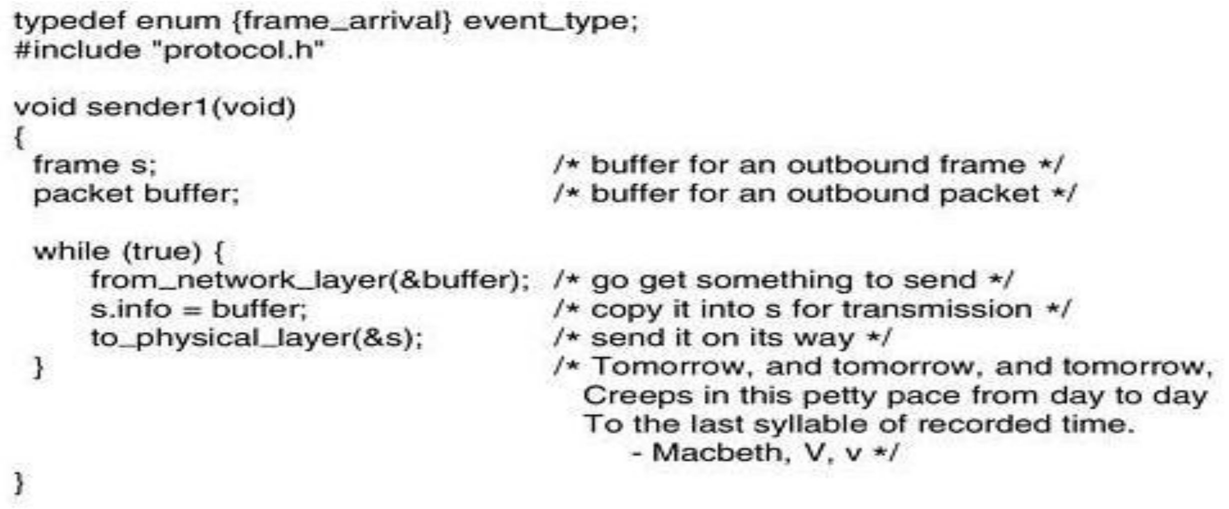
- The most sophisticated service the data link layer can provide to the network layer is connection-oriented service.
- With this service, the source and destination machines establish a connection before any data are transferred.
- Each frame sent over the connection is numbered, and the data link layer guarantees that each frame sent is indeed received.
- Furthermore, it guarantees that each frame is received exactly once and that all frames are received in the right order.
- With connectionless service, in contrast, it is conceivable that a lost acknowledgement causes a packet to be sent several times and thus received several times.
- Connection-oriented service, in contrast, provides the network layer processes with the equivalent of a reliable bit stream.
- When connection-oriented service is used, transfers go through **three** distinct phases.
- In the first phase, the connection is established by having both sides initialize variables and counters needed to keep track of which frames have been received and which ones have not.
- In the second phase, one or more frames are actually transmitted.
- In the third and final phase, the connection is released, freeing up the variables, buffers, and other resources used to maintain the connection.

ELEMENTARY DATA LINK LAYER PROTOCOLS:

An Unrestricted Simplex Protocol:

- This protocol is simple protocol.
- In this, Data are transmitted in one direction only.
- Both the transmitting and receiving network layers are always ready.
- Processing time can be ignored.
- Infinite buffer space is available.
- And best of all, the communication channel between the data link layers never damages or loses frames i.e., error free.
- This thoroughly unrealistic protocol, which we will nickname "**utopia**,"

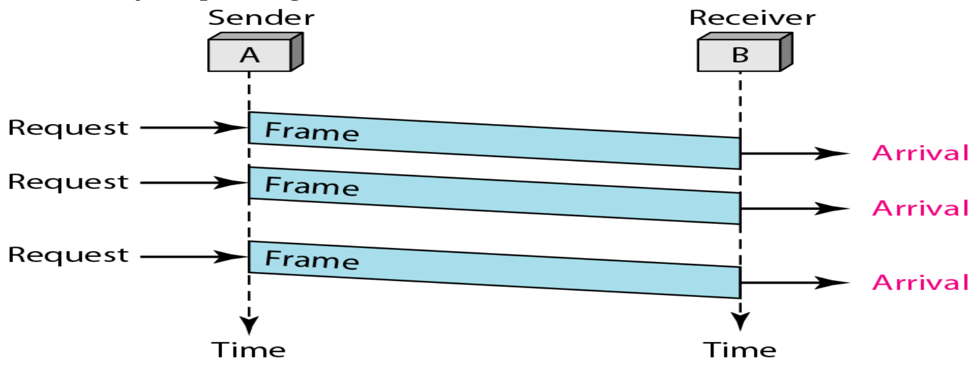
Figure: An unrestricted simplex protocol



```
typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender1(void)
{
  frame s;                          /* buffer for an outbound frame */
  packet buffer;                    /* buffer for an outbound packet */

  while (true) {
       from_network_layer(&buffer); /* go get something to send */
       s.info = buffer;             /* copy it into s for transmission */
       to_physical_layer(&s);       /* send it on its way */
  }                                 /* Tomorrow, and tomorrow, and tomorrow,
                                       Creeps in this petty pace from day to day
                                       To the last syllable of recorded time.
                                           - Macbeth, V, v */
}
```

```
void receiver1(void)
{
  frame r;
  event_type event;                    /* filled in by wait, but not used here */

  while (true) {
       wait_for_event(&event);         /* only possibility is frame_arrival */
       from_physical_layer(&r);        /* go get the inbound frame */
       to_network_layer(&r.info);      /* pass the data to the network layer */
  }
}
```

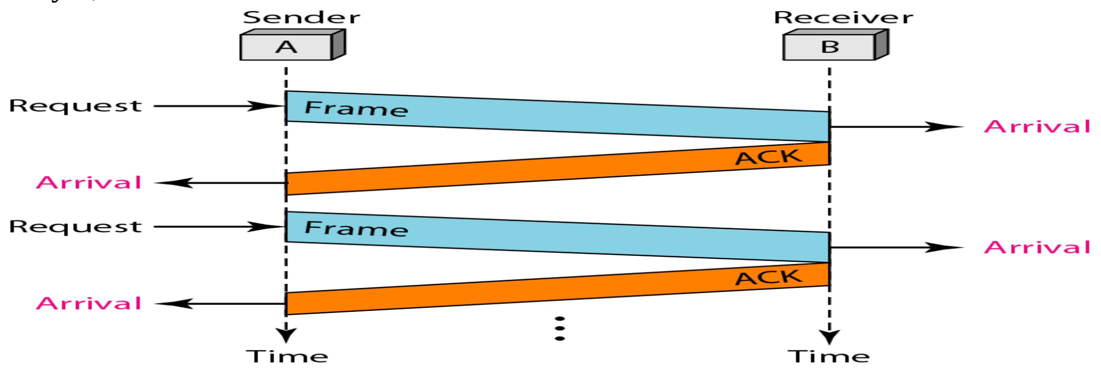
- The protocol consists of **two** distinct procedures, a sender and a receiver.
- The sender runs in the data link layer of the source machine, and the receiver runs in the data link layer of the destination machine.
- No sequence numbers or acknowledgements are used here, so MAX_SEQ is not needed.
- The only event type possible is frame_arrival.
- The sender is in an infinite while loop just pumping data out onto the line as fast as it can.
- The body of the loop consists of **three** actions:
  - go fetch a packet from the network layer,
  - construct an outbound frame using the variable s,
  - and send the frame on its way.
- Only the info field of the frame is used by this protocol, because the other fields have to do with error and flow control and there are no errors or flow control restrictions here.
- The receiver is equally simple.
- Initially, it waits for something to happen, the only possibility being the arrival of an undamaged frame.
- Eventually, the frame arrives and the procedure wait_for_event returns, with event set to frame_arrival.
- The call to from_physical_layer removes the newly arrived frame from the hardware buffer and puts it in the variable r, where the receiver code can get at it.
- Finally, the data portion is passed on to the network layer, and the data link layer settles back to wait for the next frame, effectively suspending itself until the frame arrives.



A Simplex Stop-and-Wait Protocol:

- In this protocol we will drop the most unrealistic restriction used in protocol 1: the ability of the receiving network layer to process incoming data infinitely quickly.
- The communication channel is still assumed to be error free however, and the data traffic is still simplex.

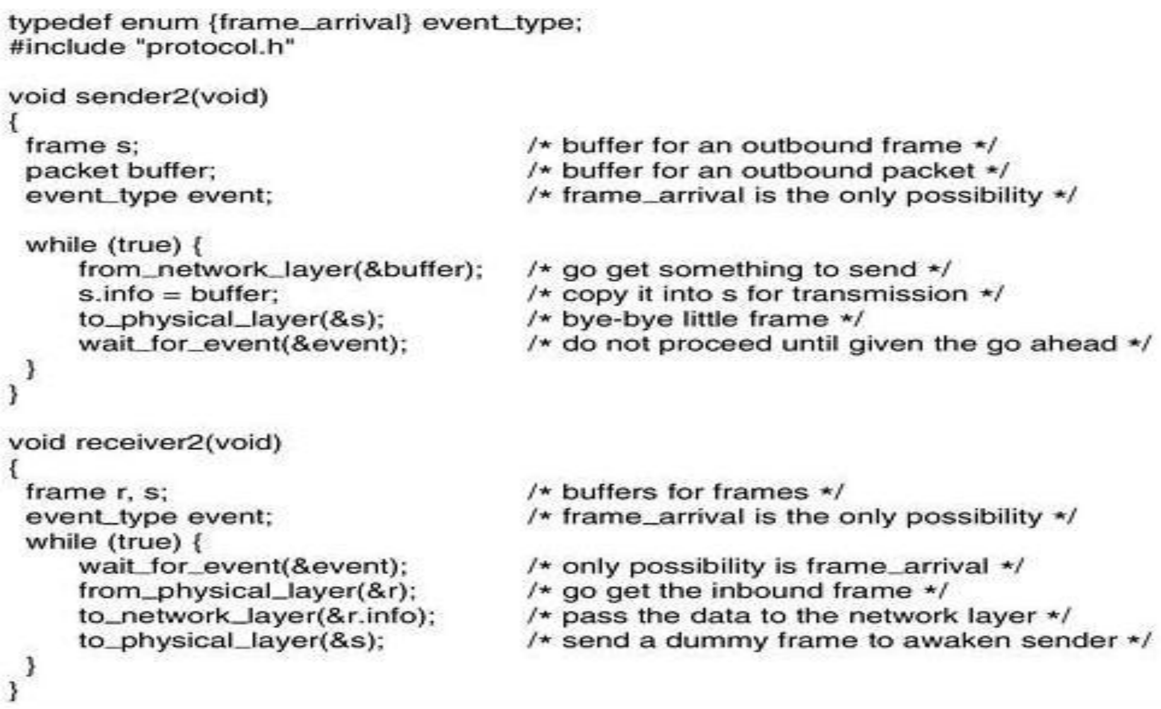
The main problem we have to deal with here is how to prevent the sender from flooding the receiver with data faster than the latter is able to process them. If we assume that no automatic buffering and queueing are done within the receiver's hardware, the sender must never transmit a new frame until the old one has been fetched by from_physical_layer, lest the new one overwrite the old one.



A more general solution to this dilemma is to have the receiver provide feedback to the sender. After having passed a packet to its network layer, the receiver sends a little dummy frame back to the sender which, in effect, gives the sender permission to transmit the next frame.

Protocols in which the sender sends one frame and then waits for an acknowledgement before proceeding are called stop-and-wait. Figure gives an example of a simplex stop-and-wait protocol.

Figure: A simplex stop-and-wait protocol.



```
typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender2(void)
{
  frame s;                          /* buffer for an outbound frame */
  packet buffer;                    /* buffer for an outbound packet */
  event_type event;                 /* frame_arrival is the only possibility */

  while (true) {
      from_network_layer(&buffer);  /* go get something to send */
      s.info = buffer;              /* copy it into s for transmission */
      to_physical_layer(&s);        /* bye-bye little frame */
      wait_for_event(&event);       /* do not proceed until given the go ahead */
  }
}

void receiver2(void)
{
  frame r, s;                       /* buffers for frames */
  event_type event;                 /* frame_arrival is the only possibility */
  while (true) {
      wait_for_event(&event);       /* only possibility is frame_arrival */
      from_physical_layer(&r);      /* go get the inbound frame */
      to_network_layer(&r.info);    /* pass the data to the network layer */
      to_physical_layer(&s);        /* send a dummy frame to awaken sender */
  }
}
```

The advantage of stop-and-wait is simplicity; each frame is checked and acknowledged before the next frame is sent.

The disadvantage is inefficiency; stop-and-wait is slow. Each frame must travel all the way to the receiver and an acknowledgement must travel all the way to the receiver and an acknowledgement must travel all the way back before the next frame can be sent. The total transmission time is more when the distance between the devices are more.

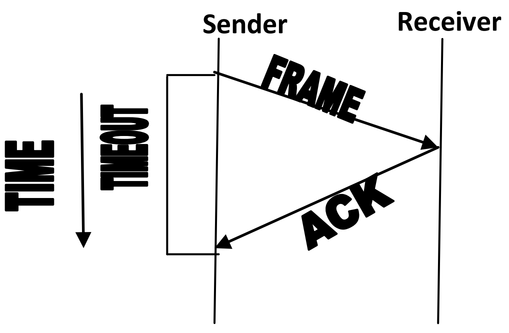
<u>A Simplex Protocol for a Noisy Channel:</u>

- In this protocol unidirectional transmission
- Noise channel
- Limited buffer
- Limited speed

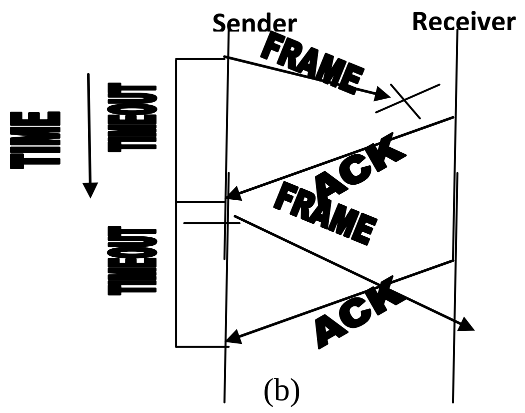
Protocols in which the sender waits for a positive acknowledgement before advancing to the next data item are often called PAR(positive Acknowledgement with retransmission) or ARQ(Automatic Repeat request).

The idea of stop-and-wait protocol over noisy channel is straightforward, after transmitting one frame, the sender waits for an acknowledgement before transmitting the next frame. If the acknowledgement does not arrive after a certain period of time, the sending time out and retransmits the original frame.

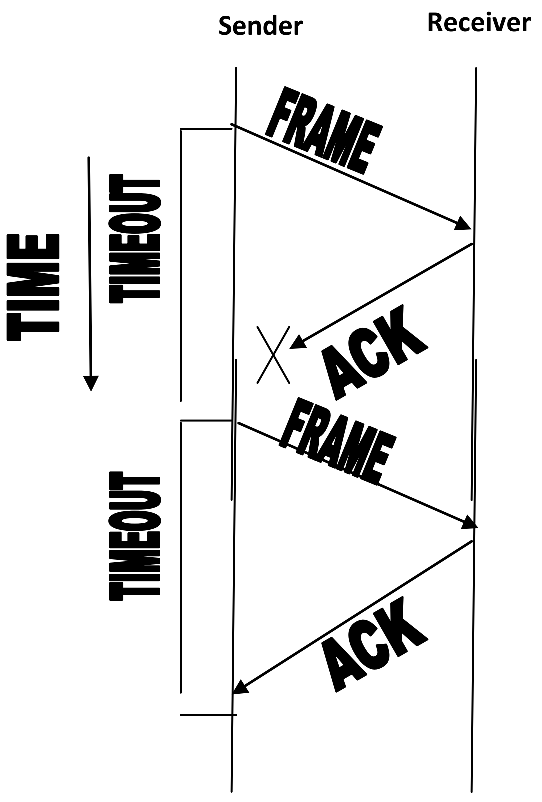
Following figure illustrates four different scenarios that result from this basic algorithm. This figure has time line, a common way to depict protocols behaviour. Figure (a) shows the situation in which the ACK is received before the time expires, figure (b) and (c) shows the situation in which the timeout fires too soon. Recall that by "lost", we mean that the frame was corrupted while in transmit, that this corruption was detected by an error code on the receiver, and that the frame was subsequently discarded.



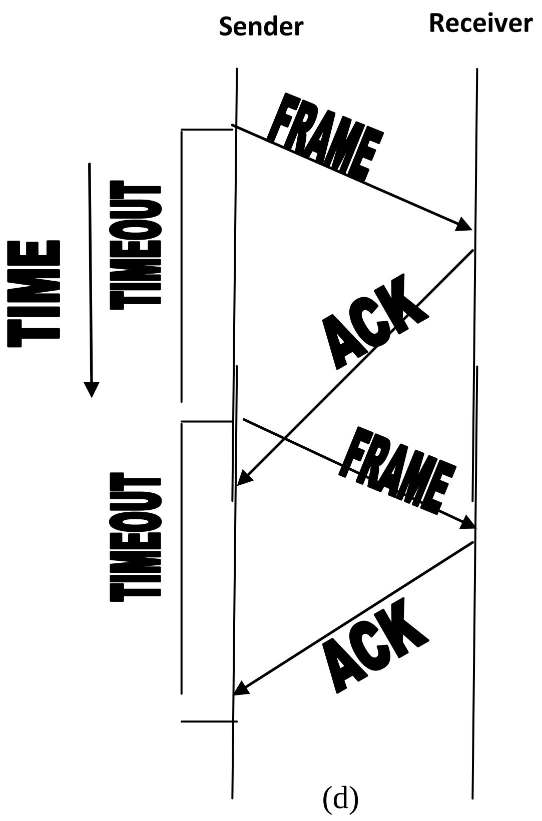
(a)



(b)



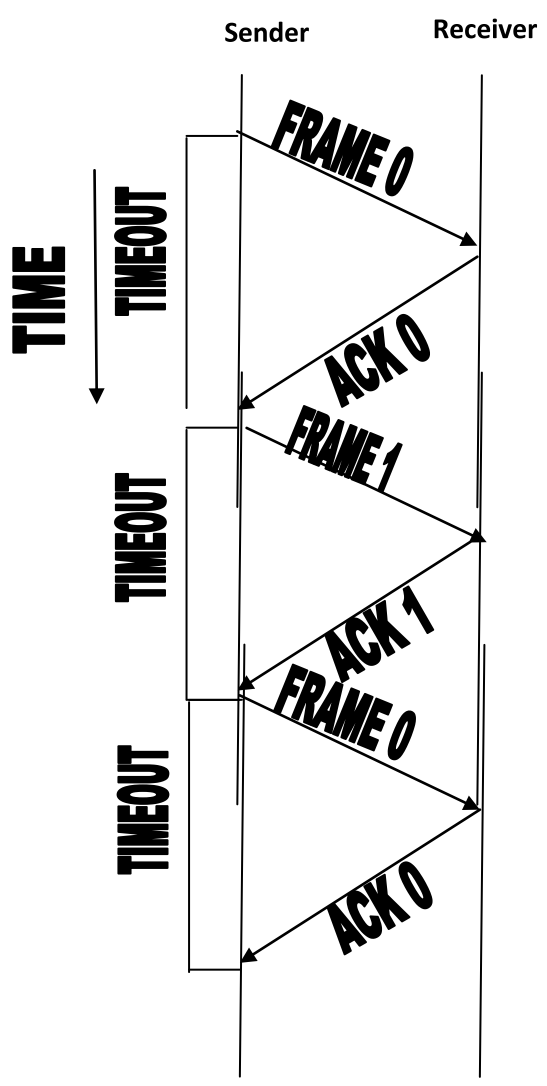
(c)

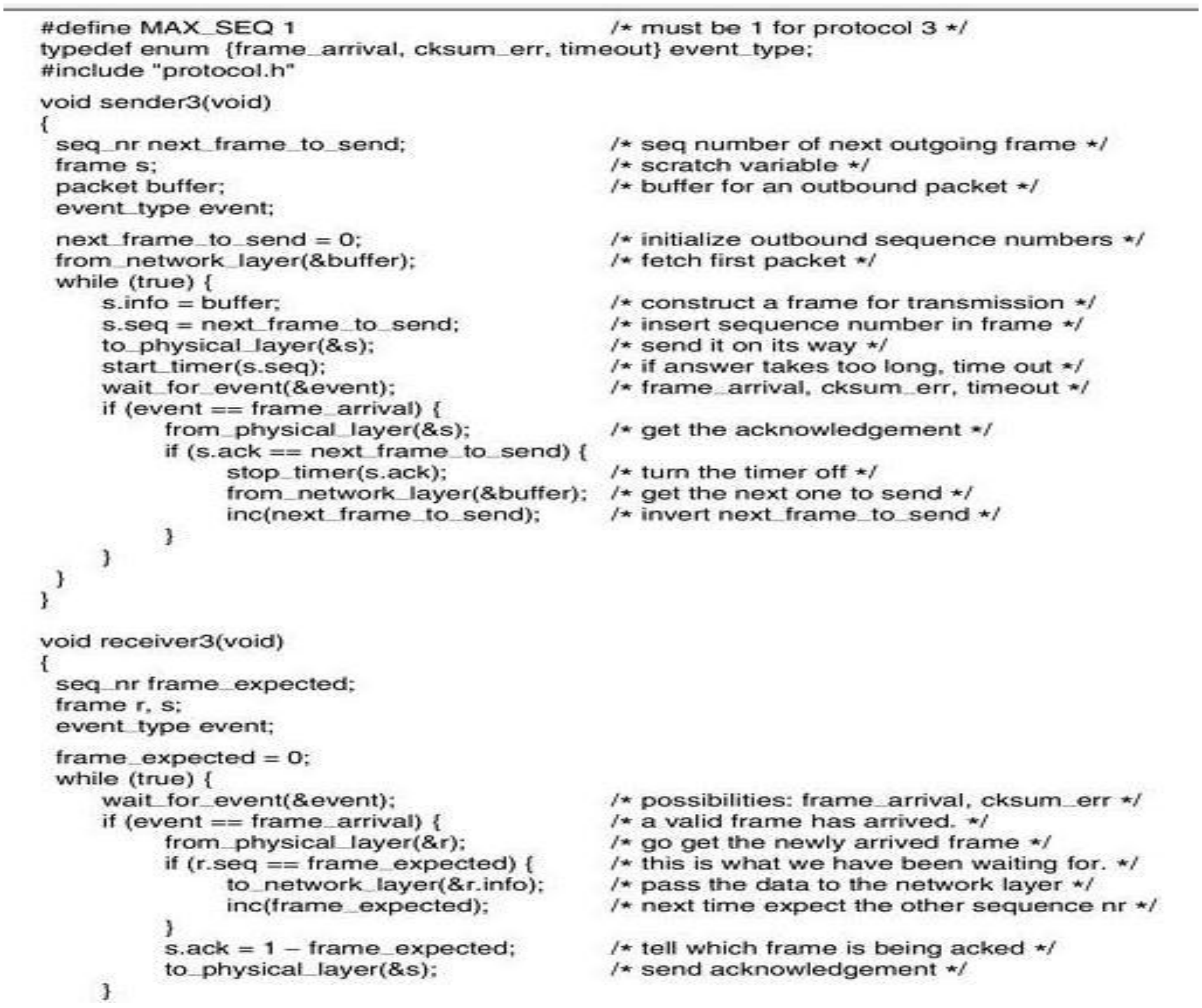


(d)

(a) The ACK is received before the timer expires (b) The original frame is lost
(c) The ACK is lost (d) The time out fires too soon

Suppose, the sender sends a frame and the receiver acknowledges it, but the acknowledgement is either lost or delayed in arriving. This situation is illustrated in figure (c) and (d). In both cases, the sender time out and transmits the original frame, but the receiver will think that it is the next frame, since it correctly received and acknowledged the first frame. This has the potential to cause duplicate copies of a frame to be delivered. To address this problem, the header for a stop-and-wait protocol usually includes a 1-bit sequence number, that is, the sequence number can take on the values 0 and 1, the sequence numbers used for each frame alternative is shown in following figure.
Thus, when the sender retransmits frame 0, the receiver can determine that it is seeing a second copy of frame 0 rather than the first copy of frame 1 and therefore can ignore it.





```
#define MAX_SEQ 1                              /* must be 1 for protocol 3 */
typedef enum  {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"

void sender3(void)
{
  seq_nr next_frame_to_send;                   /* seq number of next outgoing frame */
  frame s;                                     /* scratch variable */
  packet buffer;                               /* buffer for an outbound packet */
  event_type event;

  next_frame_to_send = 0;                      /* initialize outbound sequence numbers */
  from_network_layer(&buffer);                 /* fetch first packet */
  while (true) {
       s.info = buffer;                        /* construct a frame for transmission */
       s.seq = next_frame_to_send;             /* insert sequence number in frame */
       to_physical_layer(&s);                  /* send it on its way */
       start_timer(s.seq);                     /* if answer takes too long, time out */
       wait_for_event(&event);                 /* frame_arrival, cksum_err, timeout */
       if (event == frame_arrival) {
               from_physical_layer(&s);        /* get the acknowledgement */
               if (s.ack == next_frame_to_send) {
                       stop_timer(s.ack);      /* turn the timer off */
                       from_network_layer(&buffer); /* get the next one to send */
                       inc(next_frame_to_send);     /* invert next_frame_to_send */
               }
       }
  }
}

void receiver3(void)
{
  seq_nr frame_expected;
  frame r, s;
  event_type event;

  frame_expected = 0;
  while (true) {
       wait_for_event(&event);                 /* possibilities: frame_arrival, cksum_err */
       if (event == frame_arrival) {           /* a valid frame has arrived. */
               from_physical_layer(&r);        /* go get the newly arrived frame */
               if (r.seq == frame_expected) {  /* this is what we have been waiting for. */
                       to_network_layer(&r.info); /* pass the data to the network layer */
                       inc(frame_expected);    /* next time expect the other sequence nr */
               }
               s.ack = 1 – frame_expected;     /* tell which frame is being acked */
               to_physical_layer(&s);          /* send acknowledgement */
       }
```

Figure:. A positive acknowledgement with retransmission protocol

SLIDING WINDOW PROTOCOLS:

- In most practical situations, there is a need for transmitting data in both directions.
- One way of achieving full-duplex data transmission is to have two separate communication channels and use each one for simplex data traffic.
- If this is done, we have two separate physical circuits, each with a ”forward” channel (for data) and a ”reverse” channel (for acknowledgements).
- In both. Cases the bandwidth of the reverse channel is almost entirely wasted.
- In effect, the user is paying for two circuits but using only the capacity of one.
- A better idea is to use the same circuit for data in both directions.
- After all, in protocols 2 and 3 it was already being used to transmit frames both ways, and the reverse channel has the same capacity as the forward channel.
- In this model the data frames from A to B are intermixed with the acknowledgement frames from A to B.
- By looking at the kind field in the header of an incoming frame, the receiver can tell whether the frame is data or acknowledgement.
- When a data frame arrives, instead of immediately sending a separate control frame, the receiver restrains itself and waits until the network layer passes it the next packet.
- The acknowledgement is attached to the outgoing data frame.
- In effect, the acknowledgement gets a free ride on the next outgoing data frame.
- The technique of temporarily delaying outgoing acknowledgements so that they can be hooked onto the next outgoing data frame is known as **piggybacking**.
- The principal advantage of using piggybacking over having distinct acknowledgement frames is a better use of the available channel bandwidth.
- The ack field in the frame header costs only a few bits, whereas a separate frame would need a header, the acknowledgement, and a checksum. It rarely costs more than a few bits.
- However, piggybacking introduces a complication not present with separate acknowledgements.
- How long should the data link layer wait for a packet onto which to piggyback the acknowledgement?
- If the data link layer waits longer than the sender's timeout period, the frame will be retransmitted, defeating the whole purpose of having acknowledgements.
- If the data link layer were an oracle and could foretell the future, it would know when the next network layer packet was going to come in and could decide either to wait for it or send a separate acknowledgement immediately, depending on how long the projected wait was going to be.
- Of course, the data link layer cannot foretell the future, so it must resort to some ad hoc scheme, such as waiting a fixed number of milliseconds.
- If a new packet arrives quickly, the acknowledgement is piggybacked onto it; otherwise, if no new packet has arrived by the end of this time period, the data link layer just sends a separate acknowledgement frame.
- The next three protocols are bidirectional protocols that belong to a class called sliding window protocols.
- The three differ among themselves in terms of efficiency, complexity, and buffer requirements.
- In these, as in all sliding window protocols, each outbound frame contains a sequence number, ranging from 0 up to some maximum. The maximum is usually $2^n - 1$ so the sequence number fits exactly in an n-bit field. The stop-and-wait sliding window protocol uses $n = 1$, restricting the sequence numbers to 0 and 1, but more sophisticated versions can use arbitrary n.
- The essence of all sliding window protocols is that at any instant of time, the sender maintains a set of sequence numbers corresponding to frames it is permitted to send. These frames are said to fall within the **sending window**.
- The receiver also maintains **receiving window** corresponding to the set of frames it is permitted to accept.
- The sequence numbers within the sender’s window represent frames that have been sent or can be sent but are as yet not acknowledged.
- Whenever a new packet arrives from the network layer, it is given the next highest sequence number, and the upper edge of the window is advanced by one.
- When an acknowledgement comes in, one advances the lower edge. In this way the window continuously maintains a list of unacknowledged frames.

- Since frames currently within the sender's window may ultimately be lost or damaged in transit, the sender must keep all these frames in its memory for possible retransmission.
- Thus, if the maximum window size is n, the sender needs n buffers to hold the unacknowledged frames.
- If the window ever grows to its maximum size, the sending data link layer must forcibly shut off the network layer until another buffer becomes free.
- The receiving data link layer's window corresponds to the frames it may accept.
- Any frame falling outside the window is discarded without comment.
- When a frame whose sequence number is equal to the lower edge of the window is received, it is passed to the network layer, an acknowledgement is generated, and the window is rotated by one.
- Unlike the sender's window, the receiver's window always remains at its initial size.
- Note that a window size of 1 means that the data link layer only accepts frames in order, but for larger windows this is not so.
- The network layer, in contrast, is always fed data in the proper order, regardless of the data link layer's window size.

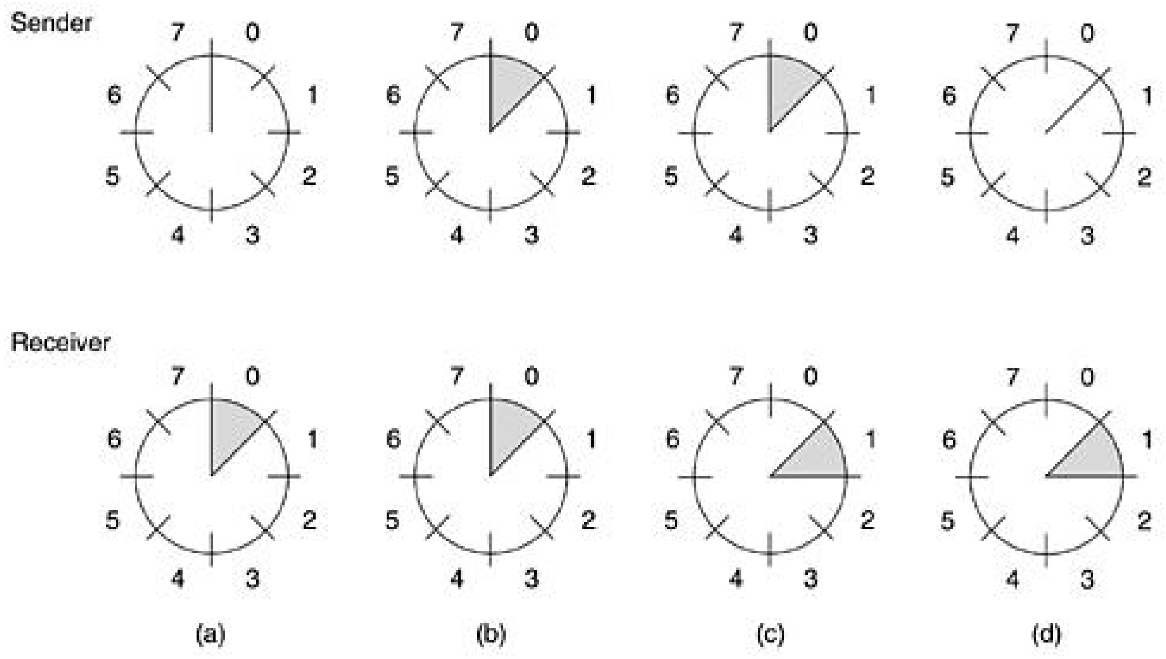


Figure: A sliding window of size=1, with a 3-bit sequence number.
(a) Initially (b) after the first frame has been sent (c) after the first frame has been received (d) after the first acknowledgement has been received

<u>A ONE BIT SLIDING WINDOW PROTOCOL:</u>

- In this sliding window protocol the maximum window size is 1.
- Such a protocol uses stop-and-wait since the sender transmits a frame and waits for its acknowledgement before sending the next one.
- Under normal circumstances, one of the two data link layers goes first and transmits the first frame.
- In the event that both data link layers start off simultaneously, a peculiar situation arises.
- The starting machine fetches the first packet from its network layer, builds a frame from it, and sends it.
- When this frame arrives, the receiving data link layer checks to see if it is a duplicate, just as in protocol 3.
- If the frame is the one expected, it is passed to the network layer and the receiver's window is slid up.
- **The acknowledgement field contains the number of the last frame received without error**.
- If this number agrees with the sequence number of the frame the sender is trying to send, the sender knows it is done with the frame stored in buffer and can fetch the next packet from its network layer.
- If the sequence number disagrees, it must continue trying to send the same frame. Whenever a frame is received, a frame is also sent back.
- Now let us examine protocol 4 to see how resilient it is to pathological scenarios.
- Assume that computer A is trying to send its frame 0 to computer B and that B is trying to send its frame 0 to A.
- Suppose that A sends a frame to B, but A's timeout interval is a little too short. Consequently, A may time out repeatedly, sending a series of identical frames, all with seq = 0 and ack = 1.
- When the first valid frame arrives at computer B, it will be accepted and frame_expected will be set to 1. All the subsequent frames will be rejected because B is now expecting frames with sequence number 1,

not 0. Furthermore, since all the duplicates have ack = 1 and B is still waiting for an acknowledgement of 0, B will not fetch a new packet from its network layer.

- After every rejected duplicate comes in, B sends A a frame containing seq = 0 and ack = 0. Eventually, one of these arrives correctly at A, causing A to begin sending the next packet. No combination of lost frames or premature timeouts can cause the protocol to deliver duplicate packets to either network layer, to skip a packet, or to deadlock.
- However, a peculiar situation arises if both sides simultaneously send an initial packet. This synchronization difficulty is illustrated by Following figure.
- In part (a), the normal operation of the protocol is shown. In (b) the peculiarity is illustrated. If B waits for A's first frame before sending one of its own, the sequence is as shown in (a), and every frame is accepted. However, if A and B simultaneously initiate communication, their first frames cross, and the data link layers then get into situation (b).
- In (a) each frame arrival brings a new packet for the network layer; there are no duplicates. In (b) half of the frames contain duplicates, even though there are no transmission errors. Similar situations can occur as a result of premature timeouts, even when one side clearly starts first. In fact, if multiple premature timeouts occur, frames may be sent three or more times.

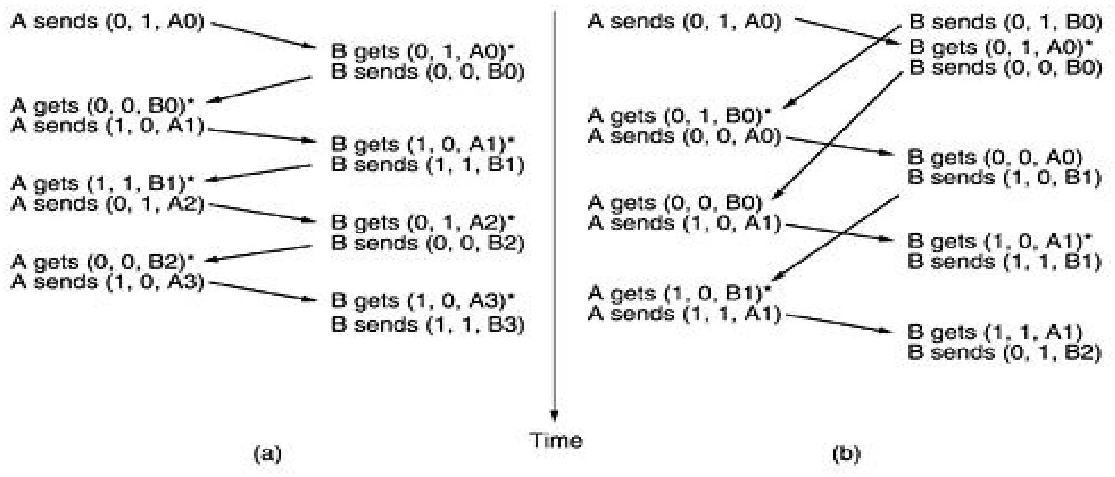


Figure:(a) Normal case (b) Abnormal case. The notation is (seq,ack,packetnumber), an asterisk indicates where a network layer accepts a packet.

<u>A Protocol Using Go Back N</u>

- **Until now we have made the tacit assumption that the transmission time required for a frame to arrive at the receiver plus the transmission time for the acknowledgement to come back is negligible**.
- Sometimes this assumption is clearly false.
- In these situations the long round-trip time can have important implications for the efficiency of the bandwidth utilization.
- Clearly, the combination of a long transit time, high bandwidth, and short frame length is disastrous in terms of efficiency.
- The problem described above can be viewed as a consequence of the rule requiring a sender to wait for an acknowledgement before sending another frame.
- If we relax that restriction, much better efficiency can be achieved.
- Basically, the solution lies in allowing the sender to transmit up to ‘w’ frames before blocking, instead of just 1.
- With an appropriate choice of w the sender will be able to continuously transmit frames for a time equal to the round-trip transit time without filling up the window.
- The need for a large window on the sending side occurs whenever the product of bandwidth x **round-trip-delay** is large.
- If the **bandwidth** is high, even for a moderate delay, the sender will exhaust its window quickly unless it has a large window.
- If the delay is high, the sender will exhaust its window even for a moderate bandwidth.
- The product of these two factors basically tells what the capacity of the pipe is, and the sender needs the ability to fill it without stopping in order to operate at peak efficiency. This technique is known as **pipelining**.
- If the channel capacity is b bits/sec, the frame size l bits, and the round-trip propagation time R sec, the time required to transmit a single frame is **l/b** sec. After the last bit of a data frame has been sent, there is

a delay of **R/2** before that bit arrives at the receiver and another delay of at least **R/2** for the acknowledgement to come back, for a total delay of **R**. In stop-and-wait the line is busy for **l/b** and idle for **R**, giving

line utilization= l / (l + bR)

- Pipelining frames over an unreliable communication channel raises some serious issues.
- What happens if a frame in the middle of a long stream is damaged or lost?
- Large numbers of succeeding frames will arrive at the receiver before the sender even finds out that anything is wrong.
- When a damaged frame arrives at the receiver, it obviously should be discarded, but what should the receiver do with all the correct frames following it?
- The receiving data link layer is obligated to hand packets to the network layer in sequence.
- Two basic approaches are available for dealing with errors in the presence of **pipelining**.
- One way, called **go back n**, is for the receiver simply to discard all subsequent frames, sending no acknowledgements for the discarded frames.
- This strategy corresponds to a receive window of size 1.
- In other words, the data link layer refuses to accept any frame except the next one it must give to the network layer.
- If the sender's window fills up before the timer runs out, the pipeline will begin to empty.
- Eventually, the sender will time out and retransmit all unacknowledged frames in order, starting with the damaged or lost one.
- This approach can waste a lot of bandwidth if the error rate is high.

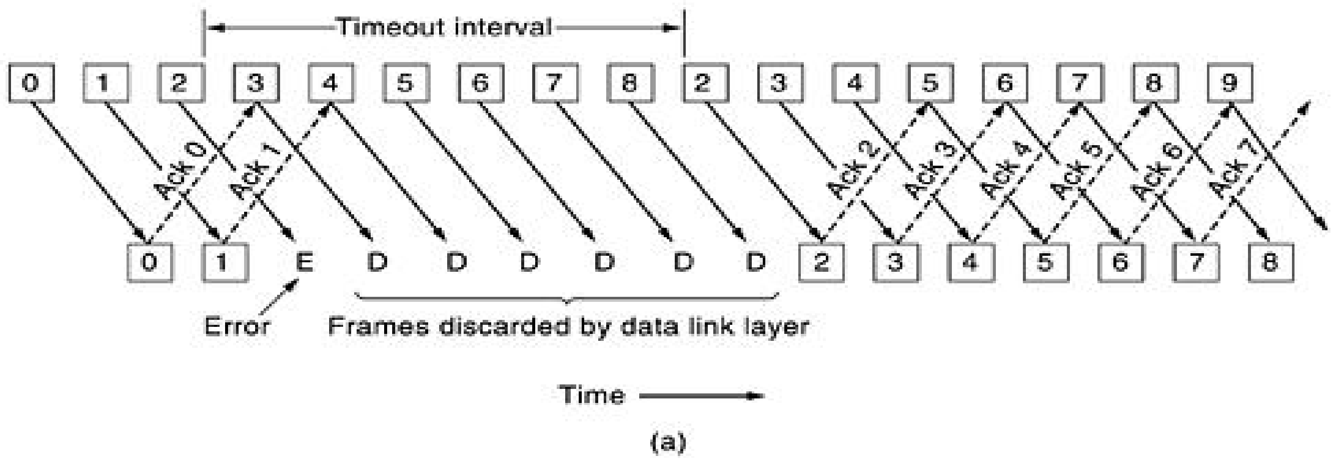


Figure: Pipelining and error recovery. Effect of an error when
(a) receiver's window size is 1

- The other general strategy for handling errors when frames are pipelined is called **selective repeat**.
- When it is used, a bad frame that is received is discarded, but good frames received after it are buffered.
- When the sender times out, only the oldest unacknowledged frame is retransmitted.
- If that frame arrives correctly, the receiver can deliver to the network layer, in sequence, all the frames it has buffered.
- Selective repeat is often combined with having the receiver send a **negative acknowledgement (NAK)** when it detects an error
- These two alternative approaches are trade-offs between bandwidth and data link layer buffer space. Depending on which resource is scarcer, one or the other can be used.

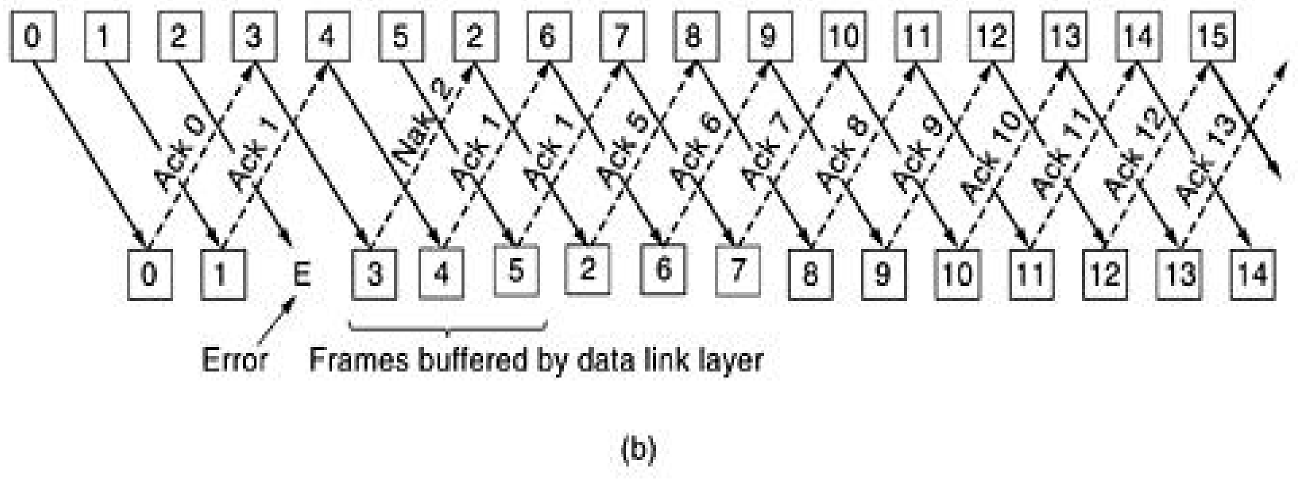
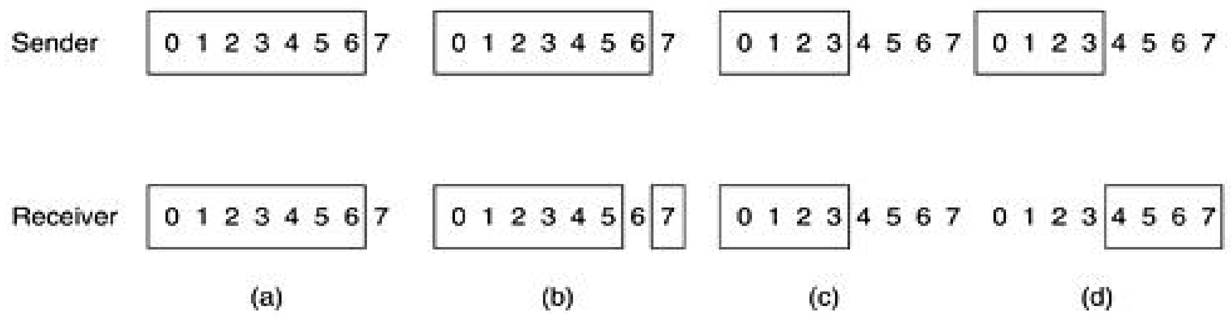


Figure: (b) receiver's window size is large.

A PROTOCOL USING SELECTIVE REPEAT:

- Protocol 5 works well if errors are rare, but if the line is poor, it wastes a lot of bandwidth on retransmitted frames.
- An alternative strategy for handling errors is to allow the receiver to accept and buffer the frames following a damaged or lost one.
- Such a protocol does not discard frames merely because an earlier frame was damaged or lost.
- In this protocol, both sender and receiver maintain a window of acceptable sequence numbers.
- The sender's window size starts out at 0 and grows to some predefined maximum, MAX_SEQ.
- The receiver's window, in contrast, is always fixed in size and equal to MAX_SEQ.
- The receiver has a buffer reserved for each sequence number within its fixed window.
- Associated with each buffer is a bit telling whether the buffer is full or empty.
- Whenever a frame arrives, its sequence number is checked and see if it falls within the window.
- If so and if it has not already been received, it is accepted and stored.
- This action is taken without regard to whether or not it contains the next packet expected by the network layer.
- Of course, it must be kept within the data link layer and not passed to the network layer until all the lower-numbered frames have already been delivered to the network layer in the correct order
- Nonsequential receive introduces certain problems not present in protocols in which frames are only accepted in order.
- Suppose that we have a 3-bit sequence number, so that the sender is permitted to transmit up to seven frames before being required to wait for an acknowledgement.
- Initially, the sender's and receiver's windows are as shown in Fig.(a).
- The sender now transmits frames 0 through 6. The receiver's window allows it to accept any frame with sequence number between 0 and 6 inclusive.
- All seven frames arrive correctly, so the receiver acknowledges them and advances its window to allow receipt of 7, 0, 1, 2, 3, 4, or 5, as shown in Fig.(b). All seven buffers are marked empty.



(a) Initial situation with a window of size seven.
(b) After seven frames have been sent and received but not acknowledged
(c) Initial situation with a window size of four.
(d) After four frames have been sent and received but not acknowledged.

- Something happened wiping out all the acknowledgements.
- The sender eventually times out and retransmits frame 0. When this frame arrives at the receiver, a check is made to see if it falls within the receiver's window.
- Unfortunately, in Fig.(b) frame 0 is within the new window, so it will be accepted.
- The receiver sends a piggybacked acknowledgement for frame 6, since 0 through 6 have been received.
- The sender is happy to learn that all its transmitted frames did actually arrive correctly, so it advances its window and immediately sends frames 7, 0, 1, 2, 3, 4, and 5.
- Frame 7 will be accepted by the receiver and its packet will be passed directly to the network layer.
- Immediately thereafter, the receiving data link layer checks to see if it has a valid frame 0 already, discovers that it does, and passes the embedded packet to the network layer.
- Consequently, the network layer gets an incorrect packet, and the protocol fails.
- The essence of the problem is that after the receiver advanced its window, the new range of valid sequence numbers overlapped the old one.
- Consequently, the following batch of frames might be either duplicates (if all the acknowledgements were lost) or new ones (if all the acknowledgements were received). The poor receiver has no way of distinguishing these two cases.

- The way out of this dilemma lies in making sure that after the receiver has advanced its window, there is no overlap with the original window.
- To ensure that there is no overlap, the maximum window size should be at most half the range of the sequence numbers, as is done in Fig.(c) and Fig.(d).
- For example, if 4 bits are used for sequence numbers, these will range from 0 to 15. Only eight unacknowledged frames should be outstanding at any instant. That way, if the receiver has just accepted frames 0 through 7 and advanced its window to permit acceptance of frames 8 through 15, it can unambiguously tell if subsequent frames are retransmissions (0 through 7) or new ones (8 through 15)
- In general, the window size for protocol 6 will be **(MAX_SEQ + 1)/2**. Thus, for 3-bit sequence numbers, the window size is four.
- An interesting question is: How many buffers must the receiver have? Under no conditions will it ever accept frames whose sequence numbers are below the lower edge of the window or frames whose sequence numbers are above the upper edge of the window.
- Consequently, the **number of buffers** needed is equal to the **window size**, not to the range of sequence numbers.
- In the above example of a 4-bit sequence number, eight buffers, numbered 0 through 7, are needed. When frame i arrives, it is put in buffer **i mod 8**.
- The **number of timers needed is equal to the number of buffers**, not to the size of the sequence space. Effectively, a timer is associated with each buffer. When the timer runs out, the contents of the buffer are retransmitted.
- In protocol 5, there is an implicit assumption that the channel is heavily loaded.
- When a frame arrives, no acknowledgement is sent immediately. Instead, the acknowledgement is piggybacked onto the next outgoing data frame.
- If the reverse traffic is light, the acknowledgement will be held up for a long period of time.
- If there is a lot of traffic in one direction and no traffic in the other direction, only MAX_SEQ packets are sent, and then the protocol blocks, which is why we had to assume there was always some reverse traffic.
- In protocol 6 this problem is fixed. After an in-sequence data frame arrives, an **auxiliary timer** is started by start_ack_timer.
- If no reverse traffic has presented itself before this timer expires, a separate acknowledgement frame is sent.
- An interrupt due to the auxiliary timer is called an ack_timeout event.
- With this arrangement, one-directional traffic flow is now possible because the lack of reverse data frames onto which acknowledgements can be piggybacked is no longer an obstacle.
- Only one auxiliary timer exists, and if start_ack_timer is called while the timer is running, it is reset to a full acknowledgement timeout interval.
- It is essential that the timeout associated with the **auxiliary timer** be appreciably **shorter than the timer used for timing out data frames**.
- This condition is required to make sure a correctly received frame is acknowledged early enough that the frame's retransmission timer does not expire and retransmit the frame.
- Protocol 6 uses a more efficient strategy than protocol 5 for dealing with errors.
- Whenever the receiver has reason to suspect that an error has occurred, it sends a negative acknowledgement (NAK) frame back to the sender.
- Such a frame is a request for retransmission of the frame specified in the NAK.
- There are two cases when the receiver should be suspicious: a damaged frame has arrived or a frame other than the expected one arrived (potential lost frame).
- To avoid making multiple requests for retransmission of the same lost frame, the receiver should keep track of whether a NAK has already been sent for a given frame.
- The variable no_nak in protocol 6 is true if no NAK has been sent yet for frame_expected.
- If the NAK gets mangled or lost, no real harm is done, since the sender will eventually time out and retransmit the missing frame anyway.
- If the wrong frame arrives after a NAK has been sent and lost, no_nak will be true and the auxiliary timer will be started. When it expires, an ACK will be sent to resynchronize the sender to the receiver's current status.

HDLC:

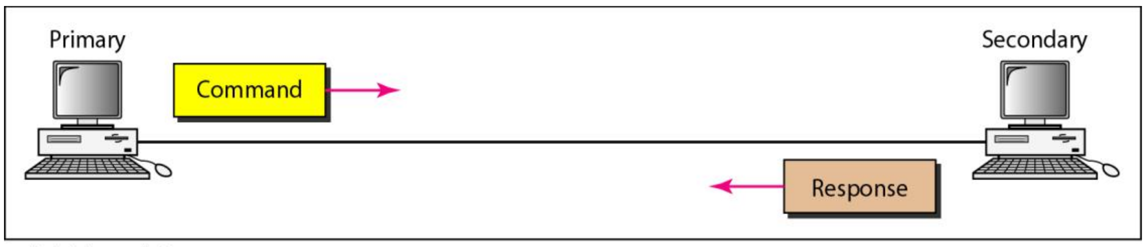
High-level Data Link Control (HDLC) is a **bit-oriented protocol** for communication over point-to-point and multipoint links. .

Configurations and Transfer Modes

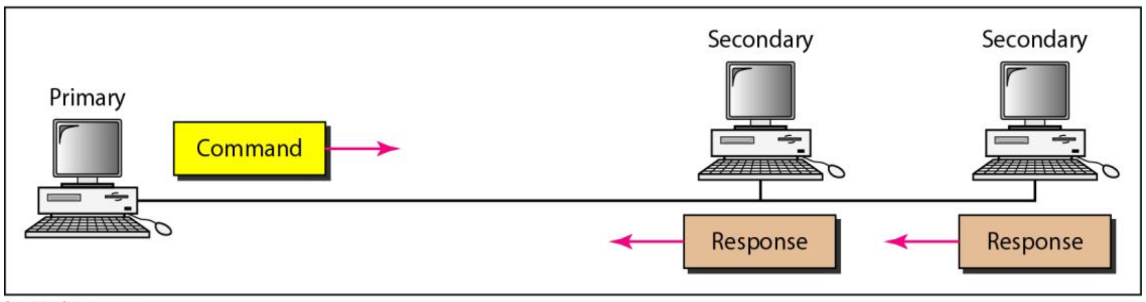
HDLC provides two common transfer modes that can be used in different configurations: **normal response mode (NRM) and asynchronous balanced mode (ABM).**

Normal Response Mode

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 multiple-point links, as shown in Figure



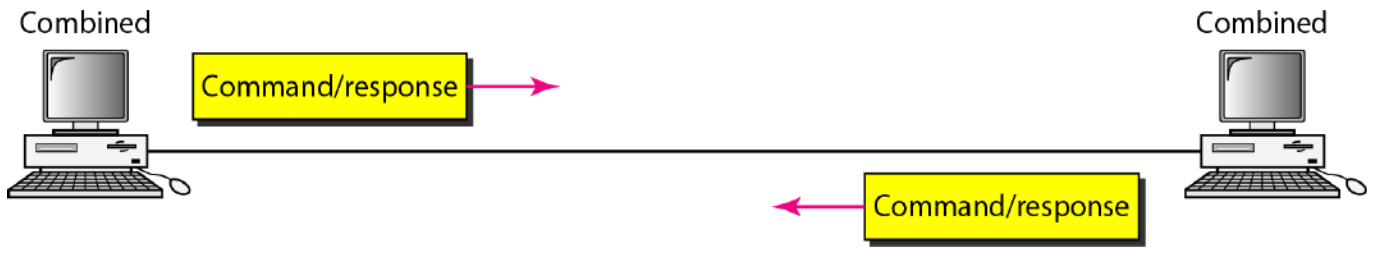
a. Point-to-point



b. Multipoint

Asynchronous Balanced Mode

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



Frames:

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

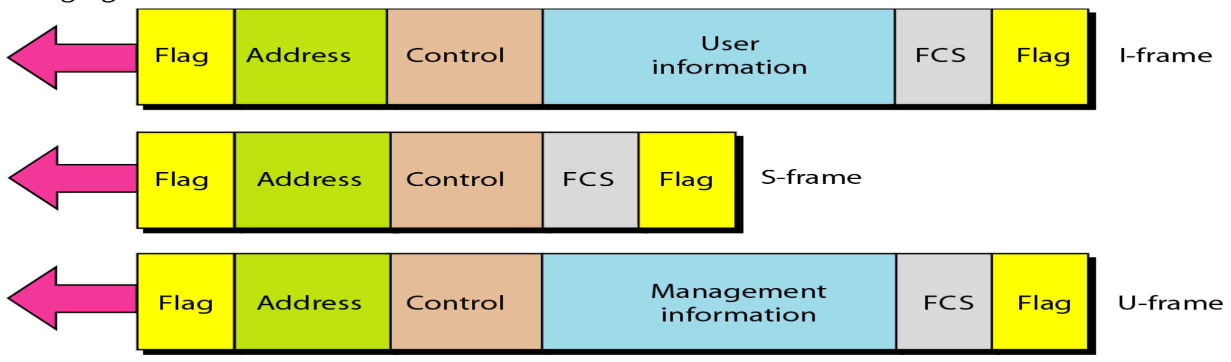


Figure: HDLC frames

Fields:

- **Flag field**. The flag field of an HDLC frame is an 8-bit sequence with the bit pattern 01111110 that identifies both the beginning and the end of a frame and serves as a synchronization pattern for the receiver.

- **Address field**. The second field of an HDLC frame contains the address of the secondary station. If a primary station created the frame, it contains a ***to address***. If a secondary creates the frame, it contains *a* ***from address***. *An address field can be* 1 byte or several bytes long, depending on the needs of the network. One byte can identify up to 128 stations. Larger networks require multiple-byte address fields. If the address field is only 1 byte, the last bit is always a 1. If the address is more than 1 byte, all bytes but the last one will end with 0; only the last will end with 1. Ending each intermediate byte with 0 indicates to the receiver that there are more address bytes to come.
- **Control field**. The control field is a 1- or 2-byte segment of the frame used for flow and error control. The interpretation of bits in this field depends on the frame type.
- **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 ITU-T CRC.

Control Field:

- The control field determines the type of frame and defines its functionality.

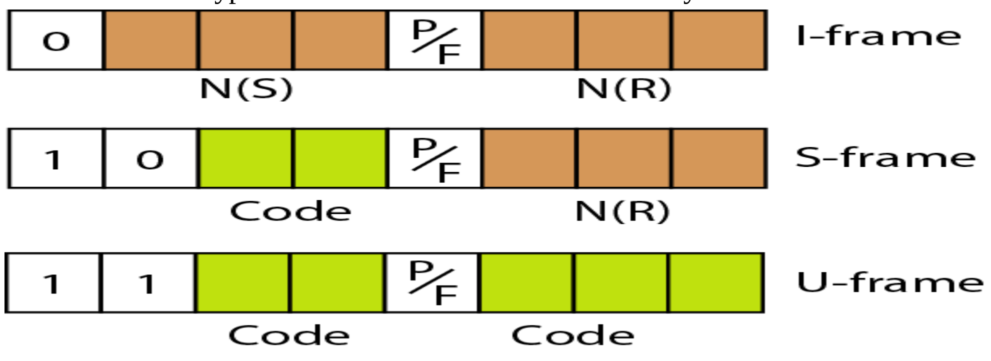
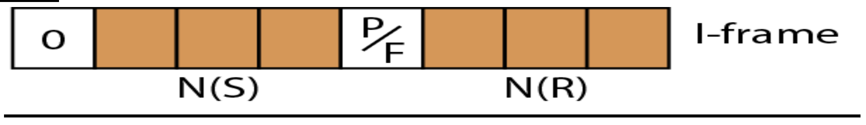


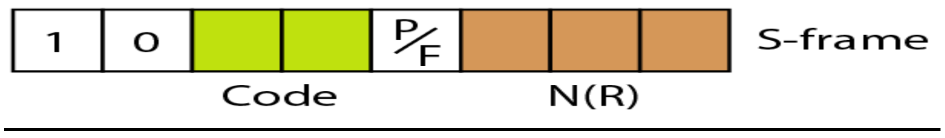
Figure: ***Control field format for the different frame types***

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. It means final when the frame is sent by a secondary to a primary.

Control Field for S-Frames:

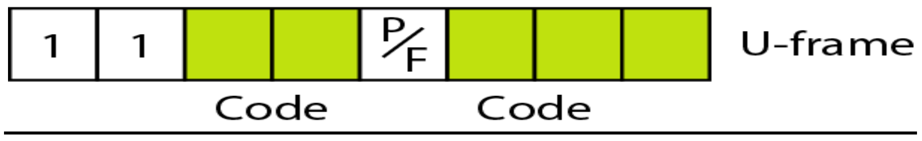


- Supervisory frames are used for flow and error control whenever piggybacking is either impossible or inappropriate.
- S-frames do not have information fields.
- If the first 2 bits of the control field is 10, this means the frame is an S-frame.
- The last 3 bits, called *N(R), corresponds to the acknowledgment number* (ACK) or negative acknowledgment number (NAK) depending on the type of S-frame.
- The 2 bits called code is 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 *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 a REJ S-frame. This is a NAK frame. 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 time 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. 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.

Code	*Command*	*Response*	*Meaning*
00 001	SNRM		Set normal response mode
11 011	SNRME		Set normal response mode, extended
11 100	SABM	**DM**	Set asynchronous balanced mode or **disconnect mode**
11 110	SABME		Set asynchronous balanced mode, extended
00 000	UI	**UI**	Unnumbered information
00 110		**UA**	**Unnumbered acknowledgment**
00 010	DISC	**RD**	Disconnect or **request disconnect**
10 000	SIM	**RIM**	Set initialization mode or **request information mode**
00 100	UP		Unnumbered poll
11 001	RSET		Reset
11 101	XID	**XID**	Exchange ID
10 001	FRMR	**FRMR**	Frame reject

Table: ***U-frame control command and response***

Example: Connection/Disconnection:

- Figure shows how U-frames can be used for connection establishment and connection release.
- Node A asks for a connection with a set asynchronous balanced mode (SABM) frame; node B gives a positive response with an unnumbered acknowledgment (UA) frame.
- After these two exchanges, data can be transferred between the two nodes.
- After data transfer, node A sends a DISC (disconnect) frame to release the connection; it is confirmed by node B responding with a VA (unnumbered acknowledgment).

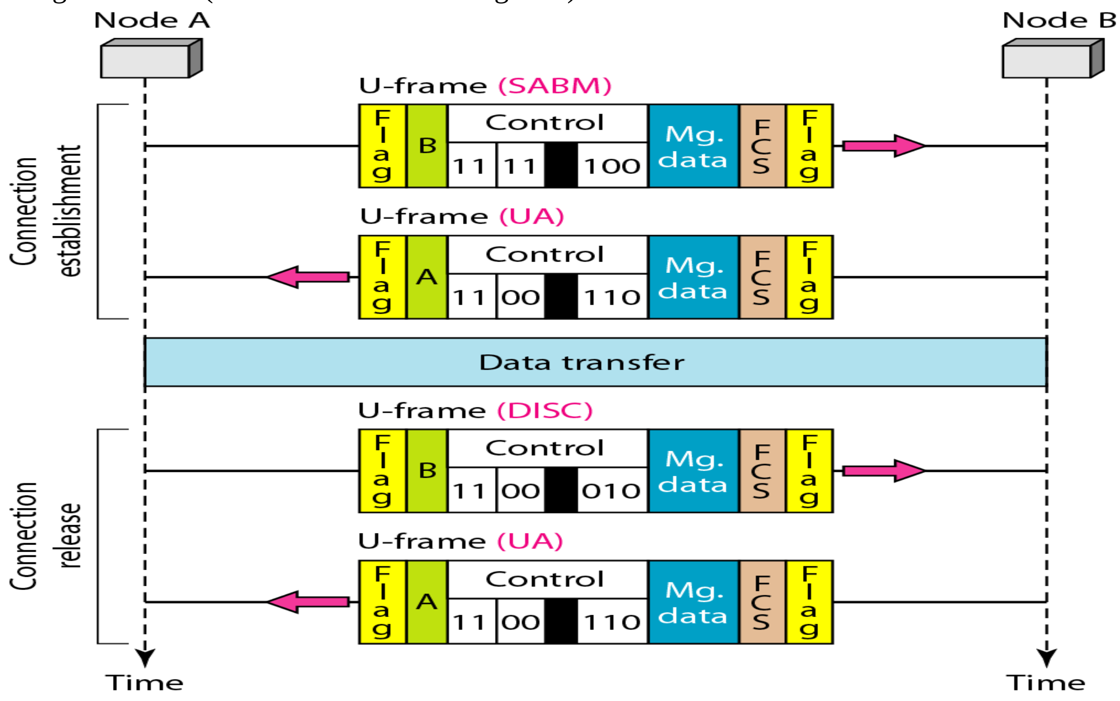


***Figure:* Example of connection and disconnection**

Example:Piggybacking without Error:

- Figure shows an exchange using piggybacking. Node A begins the exchange of information with an I-frame numbered 0 followed by another I-frame numbered 1. Node B piggybacks its acknowledgment of both frames onto an I-frame of its own. Node B's first I-frame is also numbered 0 [N(S) field] and contains a 2 in its N(R) field, acknowledging the receipt of A's frames 1 and 0 and indicating that it expects frame 2 to arrive next. Node B transmits its second and third I-frames (numbered 1 and 2) before accepting further frames from node A.

- Its N(R) information, therefore, has not changed: B frames 1 and 2 indicate that node B is still expecting A's frame 2 to arrive next. Node A has sent all its data. Therefore, it cannot piggyback an acknowledgment onto an I-frame and sends an S-frame instead. The RR code indicates that A is still ready to receive. The number 3 in the N(R) field tells B that frames 0, 1, and 2 have all been accepted and that A is now expecting frame number 3.

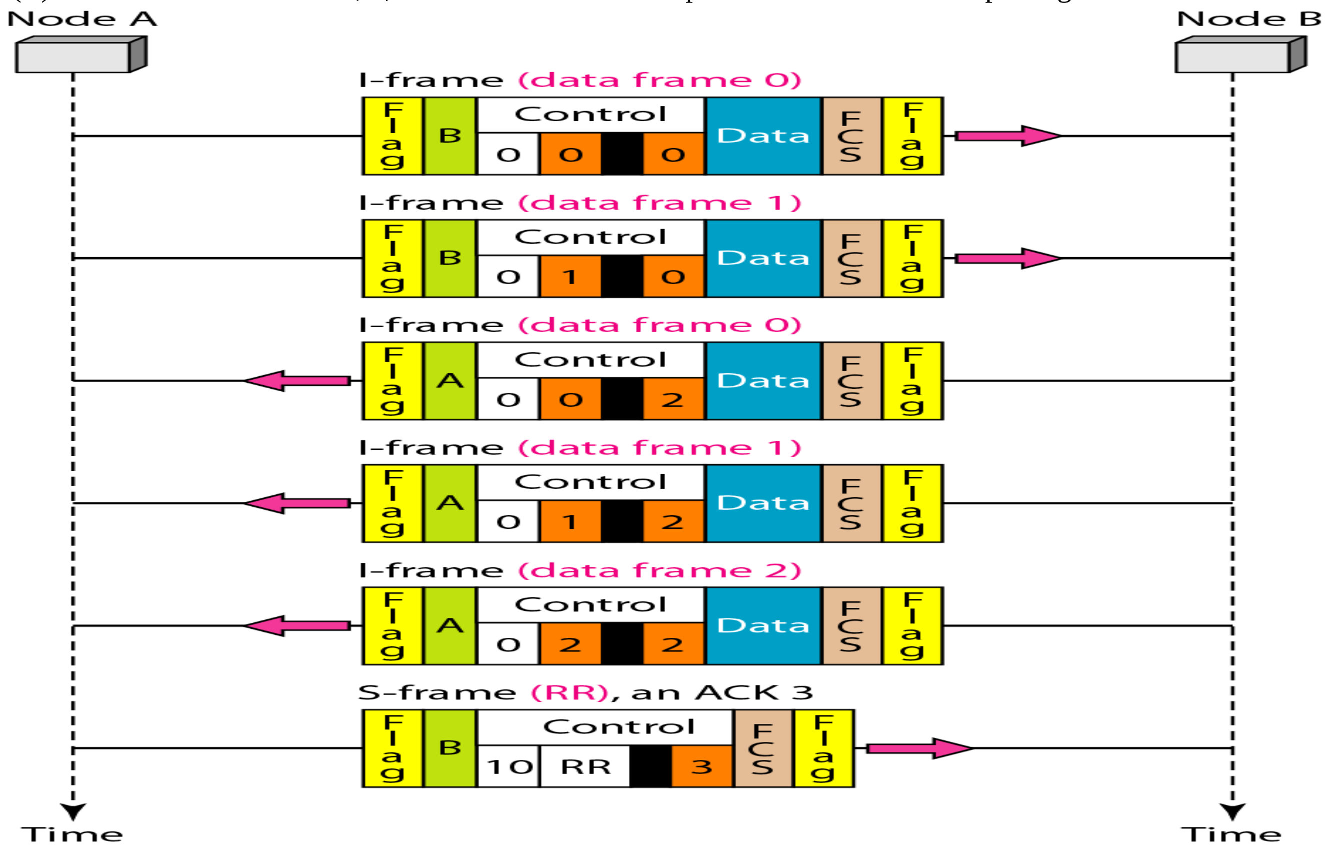

Node A
Node B
I-frame (data frame 0)
Flag B Control 0 0 0 Data FCS Flag
I-frame (data frame 1)
Flag B Control 0 1 0 Data FCS Flag
I-frame (data frame 0)
Flag A Control 0 0 2 Data FCS Flag
I-frame (data frame 1)
Flag A Control 0 1 2 Data FCS Flag
I-frame (data frame 2)
Flag A Control 0 2 2 Data FCS Flag
S-frame (RR), an ACK 3
Flag B Control 10 RR 3 FCS Flag
Time
Time

***Figure:* Example of piggybacking without error**

Example:Piggybacking with Error:

- Figure shows an exchange in which a frame is lost. Node B sends three data frames (0, 1, and 2), but frame 1 is lost. When node A receives frame 2, it discards it and sends a *REJ frame for* frame 1. Note that the protocol being used is *Go-Back-N with the special use of an REJ frame as* a NAK frame.
- The NAK frame does two things here: It confirms the receipt of frame 0 and declares that frame 1 and any following frames must be resent.
- Node B, after receiving the REJ frame, resends frames 1 and 2. Node A acknowledges the receipt by sending an RR frame (ACK) with acknowledgment number 3.

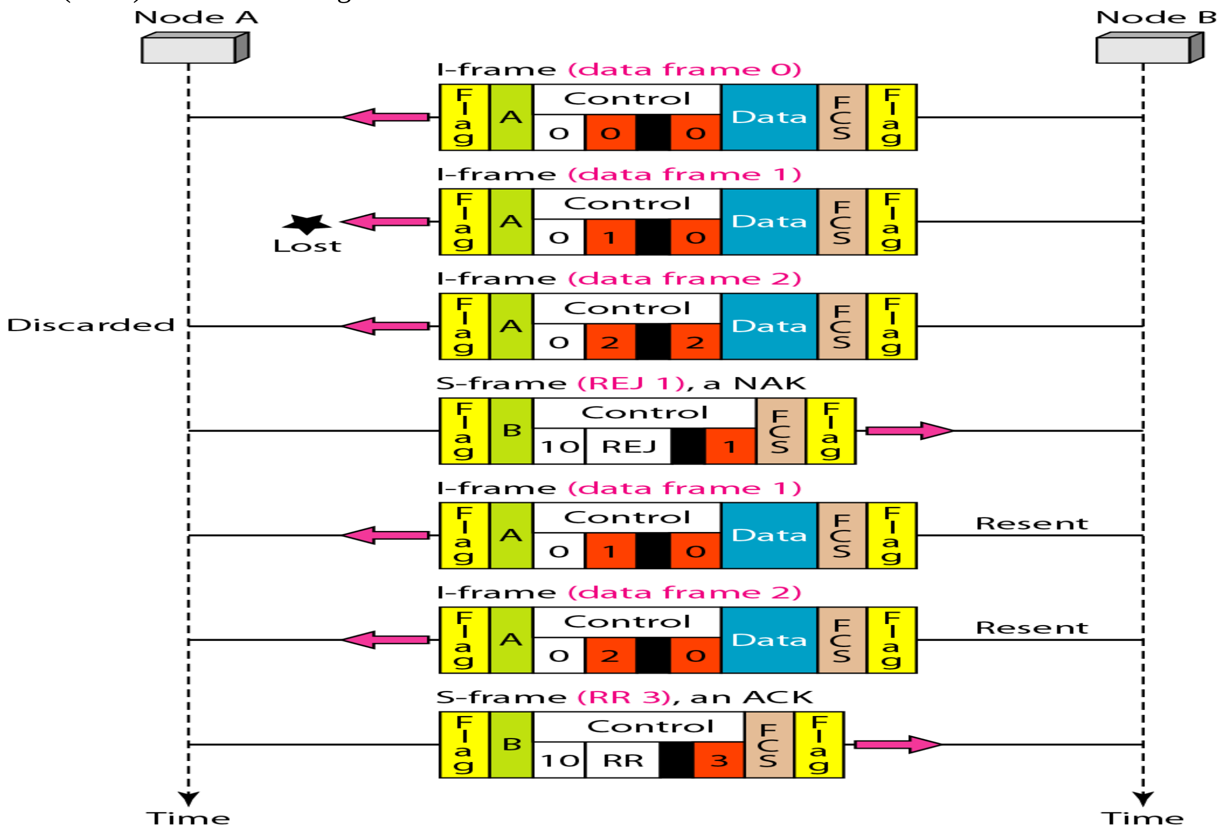

Node A
Node B
I-frame (data frame 0)
Flag A Control 0 0 0 Data FCS Flag
I-frame (data frame 1)
Lost
Flag A Control 0 1 0 Data FCS Flag
I-frame (data frame 2)
Discarded
Flag A Control 0 2 2 Data FCS Flag
S-frame (REJ 1), a NAK
Flag B Control 10 REJ 1 FCS Flag
I-frame (data frame 1)
Flag A Control 0 1 0 Data FCS Flag
Resent
I-frame (data frame 2)
Flag A Control 0 2 0 Data FCS Flag
Resent
S-frame (RR 3), an ACK
Flag B Control 10 RR 3 FCS Flag
Time
Time

Figure: *Piggybacking with Error*

POINT-TO-POINT PROTOCOL:

- Although HDLC is a general protocol that can be used for both point-to-point and multipoint configurations, 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.
- PPP provides several services:
  (a) PPP defines the format of the frame to be exchanged between devices.
  (b) PPP defines how two devices can negotiate the establishment of the link and the exchange of data.
  (c) PPP defines how network layer data are encapsulated in the data link frame.
  (d) PPP defines how two devices can authenticate each other.
  (e) PPP provides multiple network layer services supporting a variety of network layer protocols.
  (f) PPP provides connections over multiple links.
  (g) PPP provides network address configuration. This is particularly useful when a home user needs a temporary network address to connect to the Internet.
- On the other hand, to keep PPP simple, several services are missing:
  (a) PPP does not provide flow control. A sender can send several frames one after another with no concern about overwhelming the receiver.
  (b) 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.
  (c) PPP does not provide a sophisticated addressing mechanism to handle frames in a multipoint configuration.

FRAMING:

- PPP is a byte-oriented protocol.

Frame Format

- **Flag**. A PPP frame starts and ends with a 1-byte flag with the bit pattern 01111110.Although this pattern is the same as that used in HDLC, there is a big difference. PPP is a byte-oriented protocol; HDLC is a bit-oriented protocol. The flag is treated as a byte.
- **Address**. The address field in this protocol is a constant value and set to 11111111. During negotiation, the two parties may agree to omit this byte.
- **Control**. This field is set to the constant value 11000000. PPP does not provide any flow control. Error control is also limited to error detection. This means that this field is not needed at all, and again, the two parties can agree, during negotiation, to omit this byte.
- **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.

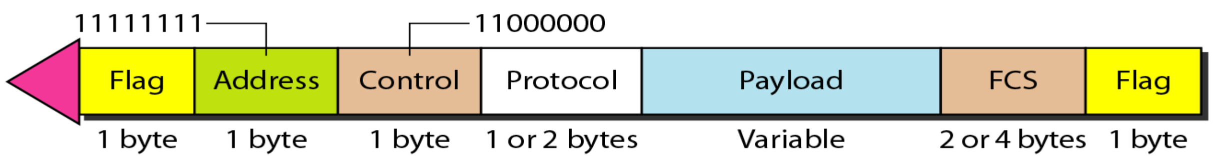


Figure: ***PPP frame format***

- **Payload field.** This field carries either the user data or other information. The data field is a sequence of bytes with the default of a maximum of 1500 bytes; but this can be changed during negotiation.
- **FCS**. The frame check sequence (FCS) is simply a 2-byte or 4-byte standard CRC.

Note: PPP is a byte-oriented protocol using byte stuffing with the escape byte 01111101.

Transition Phases:

- A PPP connection goes through phases which can be shown in a transition phase diagram

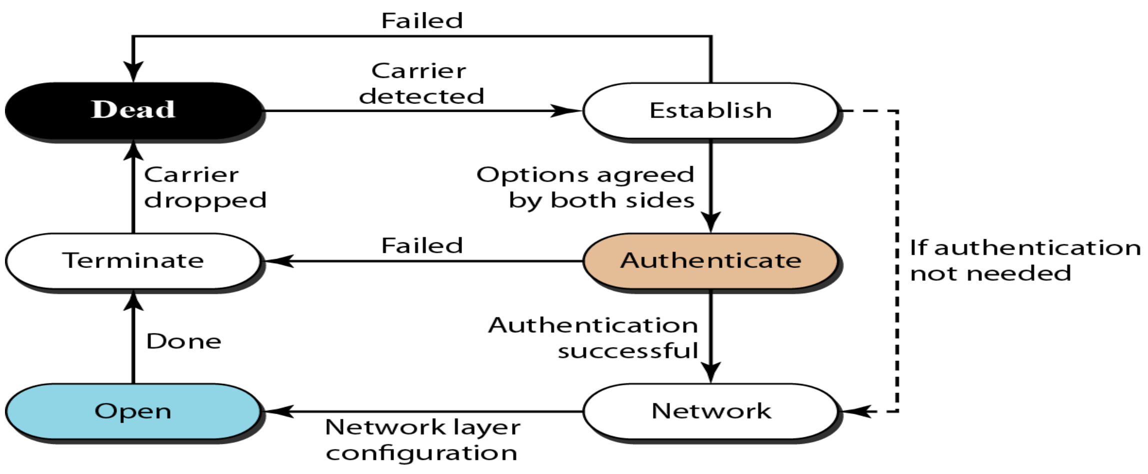

Failed
Carrier detected
Dead
Establish
Carrier dropped
Options agreed by both sides
If authentication not needed
Terminate
Failed
Authenticate
Done
Authentication successful
Open
Network
Network layer configuration

***Figure:* Transition phases**

- **Dead**. In the dead phase the link is not being used. There is no active carrier (at the physical layer) and the line is quiet.
- **Establish**. When one of the nodes starts the communication, the connection goes into this phase. In this phase, options are negotiated between the two parties. If the negotiation is successful, the system goes to the authentication phase or directly to the networking phase. The **link control protocol** packets, are used for this purpose. Several packets may be exchanged here.
- **Authenticate**. The authentication phase is optional; the two nodes may decide, during the establishment phase, not to skip this phase. However, if they decide to proceed with authentication, they send several authentication packets. If the result is successful, the connection goes to the networking phase; otherwise, it goes to the termination phase.
- **Network**. In the network phase, negotiation for the network layer protocols takes place. PPP specifies that two nodes establish a network layer agreement before data at the network layer can be exchanged. The reason is that PPP supports multiple protocols at the network layer. If a node is running multiple protocols simultaneously at the network layer, the receiving node needs to know which protocol will receive the data.
- **Open**. In the open phase, data transfer takes place. When a connection reaches this phase, the exchange of data packets can be started. The connection remains in this phase until one of the endpoints wants to terminate the connection.
- **Terminate**. In the termination phase the connection is terminated. Several packets are exchanged between the two ends for house cleaning and closing the link.

MULTIPLEXING:

- Although PPP is a data link layer protocol, PPP uses another set of other 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.

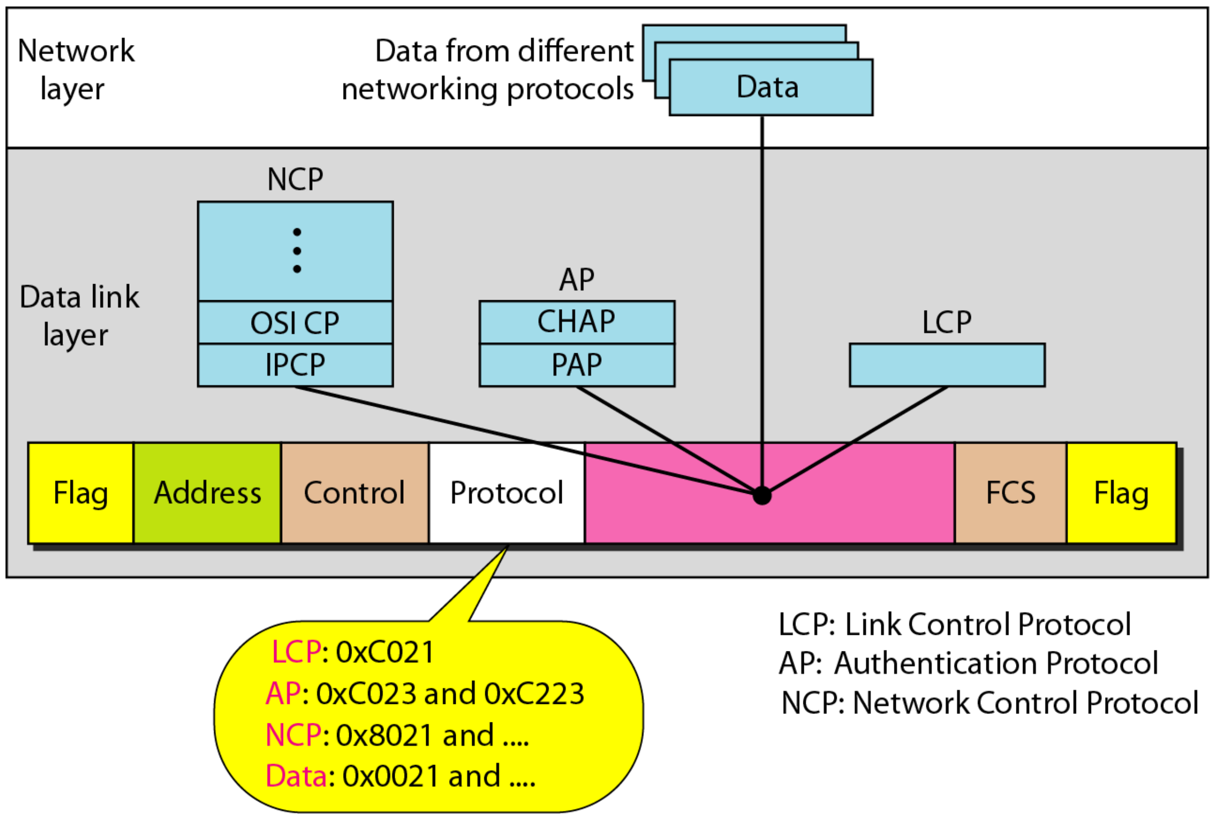


***Figure:* Multiplexing in PPP**

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.
- All LCP packets are carried in the payload field of the PPP frame with the protocol field set to C021 in hexadecimal.
- The code field defines the type of LCP packet. There are 11 types of packets

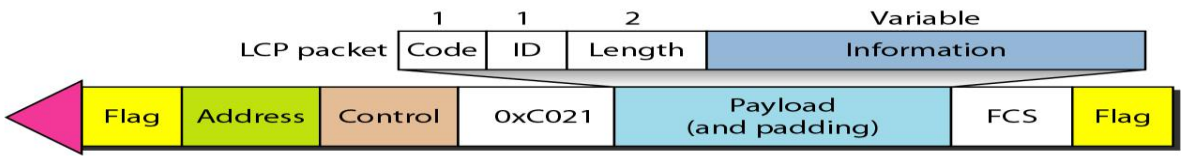


Figure : LCP packet encapsulated in a frame

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*

- There are three categories of packets.
- The first category, comprising the first four packet types, is used for link configuration during the establish phase.
- The second category, comprising packet types 5 and 6, is used for link termination during the termination phase.
- The last five packets are used for link monitoring and debugging.
- The ID field holds a value that matches a request with a reply.
- One endpoint inserts a value in this field, which will be copied into the reply packet.
- The length field defines the length of the entire LCP packet.
- The information field contains information, such as options, needed for some LCP packets.
- There are many options that can be negotiated between the two endpoints.
- Options are inserted in the information field of the configuration packets.
- In this case, the information field is divided into three fields: option type, option length, and option data. We list some of the most common options in Table

Option	*Default*
Maximum receive unit (payload field size)	1500
Authentication protocol	None
Protocol field compression	Off
Address and control field compression	Off

<u>Authentication Protocols:</u>

- 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:

1. The user who wants to access a system sends an authentication identification (usually the user name) and a password.
2. The system checks the validity of the identification and password and either accepts or denies connection.
3. Figure shows the three types of packets used by PAP and how they are actually exchanged.
4. When a PPP frame is carrying any PAP packets, the value of the protocol field is OxC023.
5. The three PAP packets are **authenticate-request, authenticate-ack, and authenticate-nak**.
6. The first packet is used by the user to send the user name and password.
7. The second is used by the system to allow access.
8. The third is used by the system to deny access.

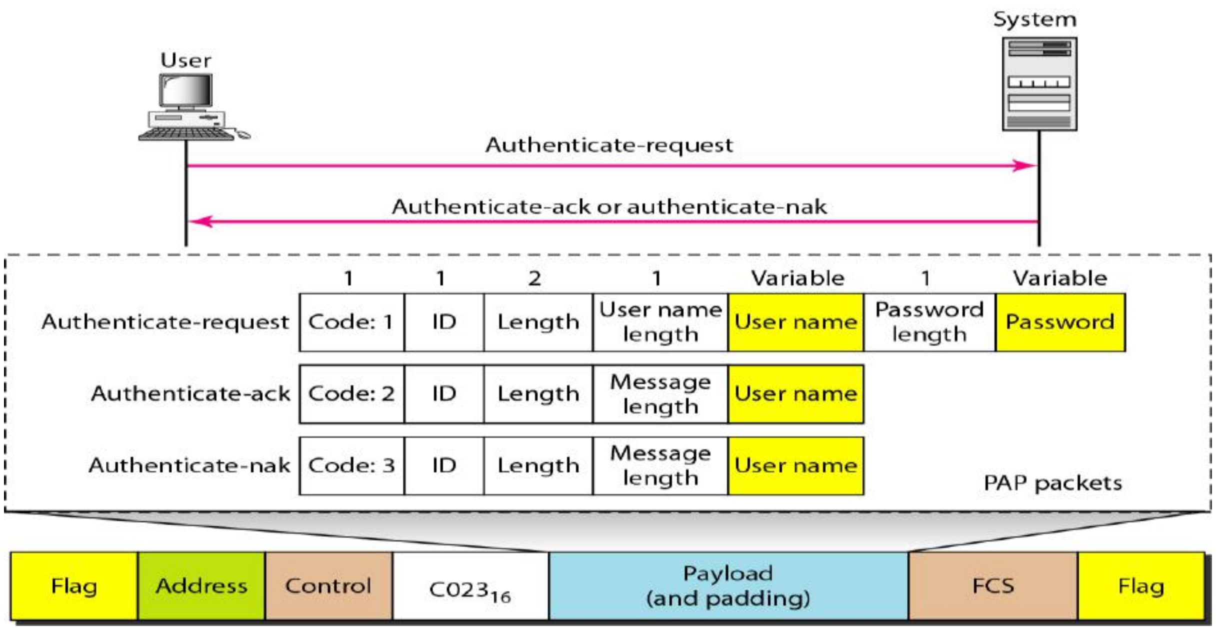


***Figure:* PAP packets encapsulated in a PPP frame**

CHAP The Challenge Handshake Authentication Protocol (CHAP):

- It is a three-way hand-shaking authentication protocol that provides greater security than PAP.
- In this method, the password is kept secret; it is never sent online.

1. The system sends the user a challenge packet containing a challenge value, usually a few bytes.
2. 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.
3. The system does the same. It applies the same function to the password of the user 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; otherwise, it is denied. CHAP is more secure than PAP, especially if the system continuously changes the challenge value. Even if the intruder learns the challenge value and the result, the password is still secret. Figure shows the packets and how they are used.

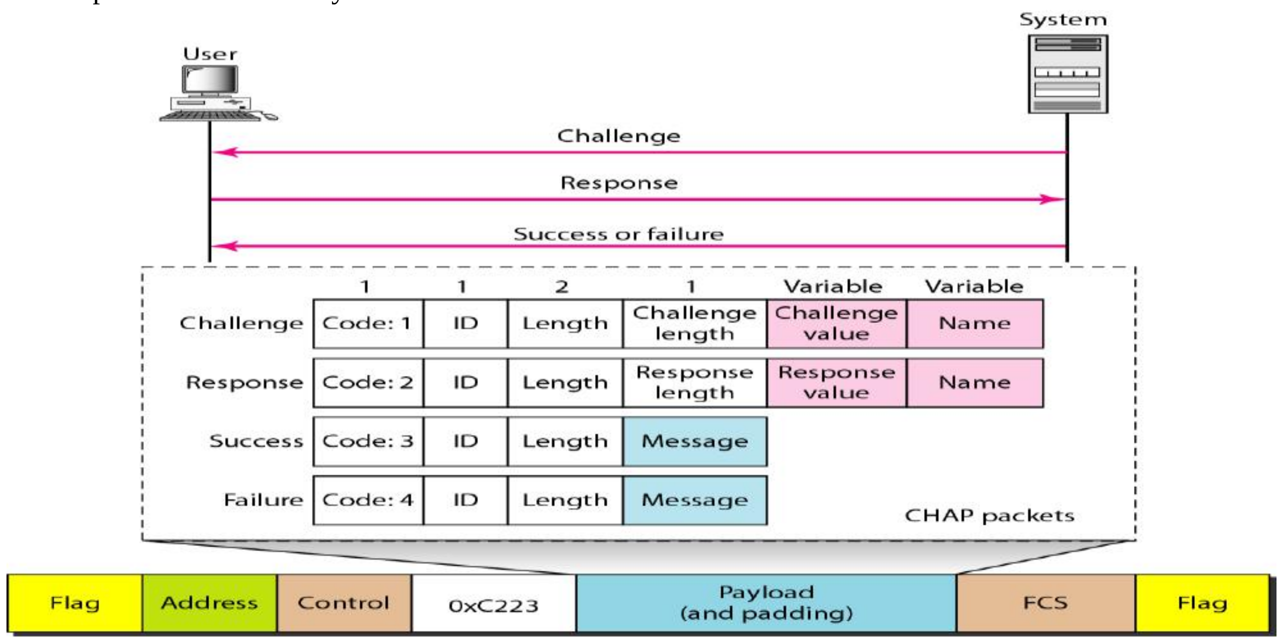


Figure : CHAP packets encapsulated in a PPP frame

- CHAP packets are encapsulated in the PPP frame with the protocol value C223 in hexadecimal.
- There are **four CHAP packets: challenge, response, success, and failure.**
- The first packet is used by the system to send the challenge value.
- The second is used by the user to return the result of the calculation.
- The third is used by the system to allow access to the system.
- The fourth is used by the system to deny access to the system.

Network Control Protocols:

- 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.
- **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.

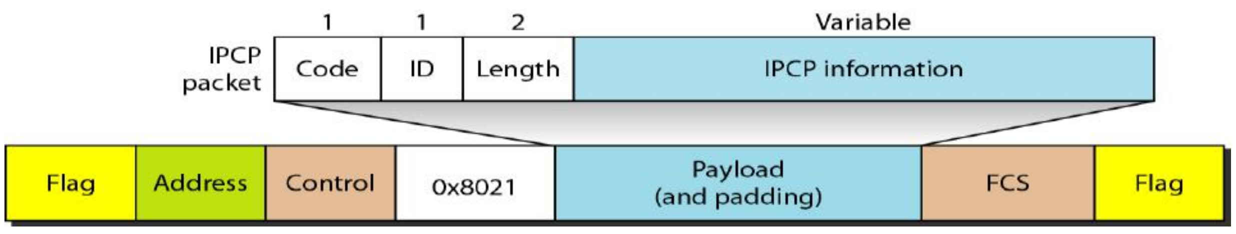


Figure: IPCP packet encapsulated in PPP frame

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

Table: Code value for IPCP packets

- 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.
- The value of the code and the format of the packets for these other protocols are the same as shown in Table.

<u>DATA FROM THE NETWORK LAYER:</u>

- 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.
- For example, if PPP is carrying data from the IP network layer, the field value is 0021. If PPP is carrying data from the OSI network layer, the value of the protocol field is 0023, and so on. Figure shows the frame for IP.

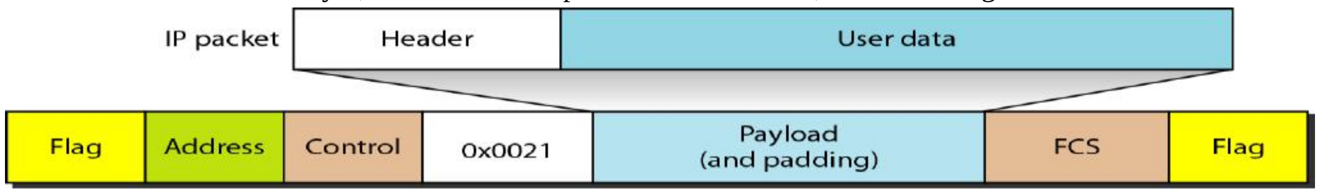


Figure: IP datagram encapsulated in a PPP frame

<u>Multilink PPP</u>:

- 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 Figure.
- To show that the actual PPP frame is carrying a fragment of a logical PPP frame, the protocol field is set to Ox003d. 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.

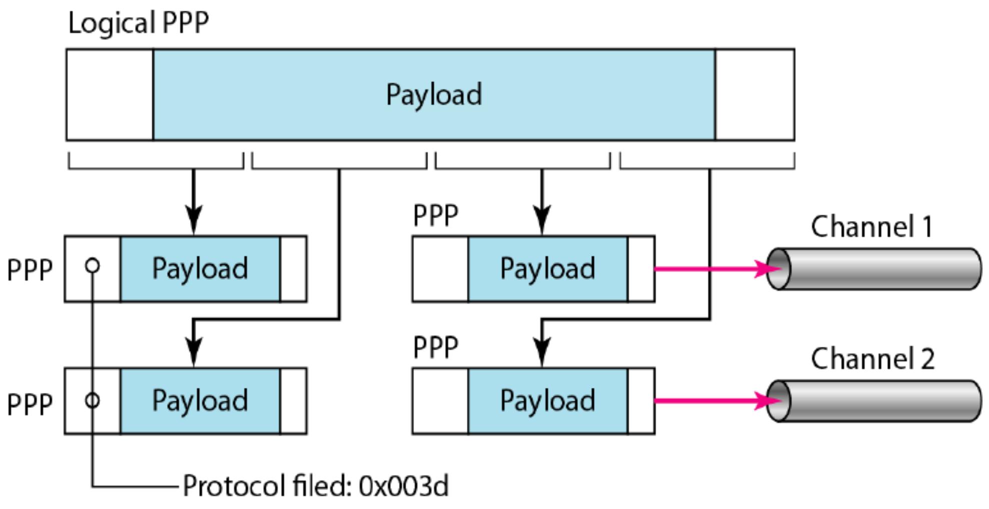


Figure: Multilink PPP