COMPUTER NETWORKS

UNIT-3

Syllabus: The Data Link Layer - Data Link Layer Design Issues, Services Provided to the Network Layer - Framing - Error Control - Flow Control, Error Detection and Correction - Error-Correcting Codes - Error Detecting Codes, Elementary Data Link Protocols- A Utopian Simplex Protocol-A Simplex Stop and Wait Protocol for an Error free channel-A Simplex Stop and Wait Protocol for a Noisy Channel, Sliding Window Protocols-A One Bit Sliding Window Protocol-A Protocol Using Go-Back-NA Protocol Using Selective Repeat

~~~~~~

DESIGN ISSUES:

- The function of the data link layer is to take services from the physical layer and provide services to the network layer.
- The two main functions of the data link layer are **data link control** and **media access control.**
- Data link control functions include framing, flow and error control, and software implemented protocols that provide smooth and reliable transmission of frames between nodes.

SERVICES PROVIDED TO NETWORK LAYER:

- The primary responsibility of Data link layer is to provide services to the network layer. The one of the major service provided is the transferring the data from network layer on the source machine to the network layer on destination machine.
- The data link layer has a number of specific functions it can carry out. These functions include

  1. Providing a well-defined service interface to the network layer.
  2. Dealing with transmission errors.
  3. Regulating the flow of data so that slow receivers are not swamped by fast senders.

1. Services Provided to the Network Layer:

- 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 figure. The actual transmission follows the path of figure, but it is easier to think in terms of two data link layer processes communicating using a data link protocol.
~~~~~~

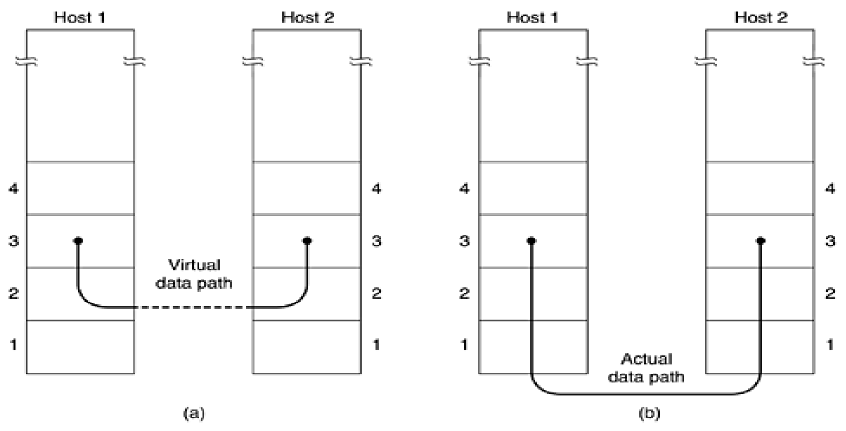


Fig:3.16 Services Provided to the Network Layer

- The three major types of services offered by data link layer are:
  1. Unacknowledged connectionless service.
  2. Acknowledged connectionless service.
  3. Acknowledged connection oriented service.

1. Unacknowledged Connectionless Service

- In this type of service source machine sends frames to destination machine but the destination machine does not send any acknowledgement of these frames back to the source. Hence it is called unacknowledged service.
- There is no connection establishment between source and destination machine before data transfer or release after data transfer. Therefore it is known as connectionless service.
- There is no error control i.e. if any frame is lost due to noise on the line, no attempt is made to recover it.
- This type of service is used when error rate is low and It is suitable for real time traffic such as speech.

2. Acknowledged Connectionless Service

- In this service, neither the connection is established before the data transfer nor is it released after the data transfer between source and destination.

- When the sender sends the data frames to destination, destination machine sends back the acknowledgement of these frames.
- This type of service provides additional reliability because source machine retransmit the frames if it does not receive the acknowledgement of these frames within the specified time.
- This service is useful over unreliable channels, such as wireless systems.

3. Acknowledged Connection – Oriented Service

- This service is the most sophisticated service provided by data link layer to network layer.
- It is connection-oriented. It means that connection is establishment between source & destination before any data is transferred.
- In this service, data transfer has three distinct phases:-Connection establishment, Actual data transfer, Connection release.
- Here, each frame being transmitted from source to destination is given a specific number and is acknowledged by the destination machine.
- All the frames are received by destination in the same order in which they are send by the source.

FRAMING:

- The DLL translates the physical layer's raw bit stream into discrete units (messages) called ***frames***.
- 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.
- When a message is carried in one very large frame, even a single-bit error would require the retransmission of the whole message. When a message is divided into smaller frames, a single-bit error affects only that small frame.

Types of framing:

- **Fixed-Size Framing**
- **Variable-Size Framing**

1. **Fixed-Size Framing:**
   - 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.
2. **Variable-Size Framing:**
   - Variable-size framing is used in local- area networks.
   - In variable-size framing, we need a way to define the end of the frame and the beginning of the next.
   - Four approaches were used for this purpose:
     - **Character count.**
     - **Flag bytes with byte stuffing.**
     - **Starting and ending flags, with bit stuffing.**
     - **Physical layer coding violations.**

Character Count:

The first framing method uses a field in the header to specify the number of characters in the frame. When the data link layer at the destination sees the character count, it knows how many characters follow and hence where the end of the frame is. This technique is shown in following Fig. 3.1 (a) for four frames of sizes 5, 5, 8, and 8 characters, respectively.

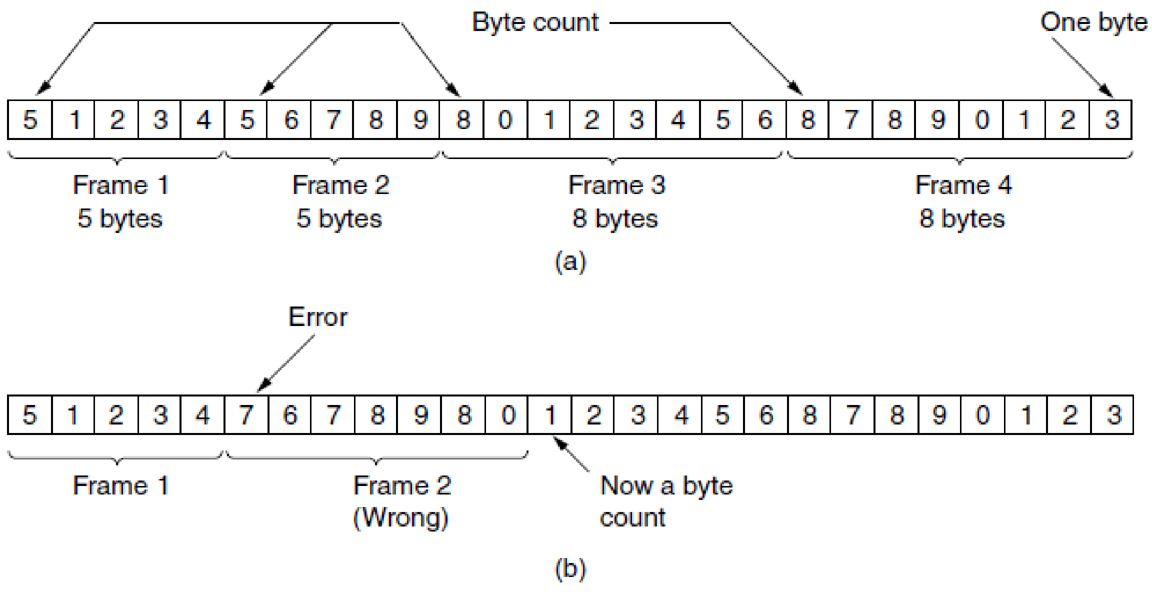


Figure 3-1. *A byte stream. (a) Without errors. (b) With one error.*

The trouble with this algorithm is that the count can be garbled by a transmission error. For example, if the character count of 5 in the second frame of Fig. 3.1(b) becomes a 7, the destination will get out of synchronization and will be unable to locate the start of the next frame. Even if the checksum is incorrect so the destination knows that the frame is bad, it still has no way of telling where the next frame starts. Sending a frame back to the source asking for a retransmission does not help either, since the destination does not know how many characters to skip over to get to the start of the retransmission. For this reason, the character count method is rarely used anymore.

Flag bytes with byte stuffing:

- *Character-oriented framing was popular when only text was exchanged by the data link layers.*

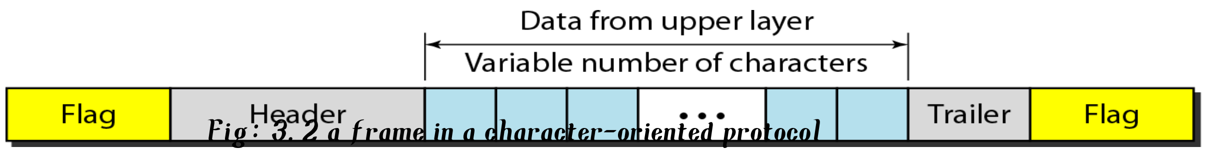


Fig: 3.2 a frame in a character-oriented protocol

- *If we send other types of information such as graphs, audio, and video, there will be a chance of this data to be matched with the flag data, so what happens is 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.*

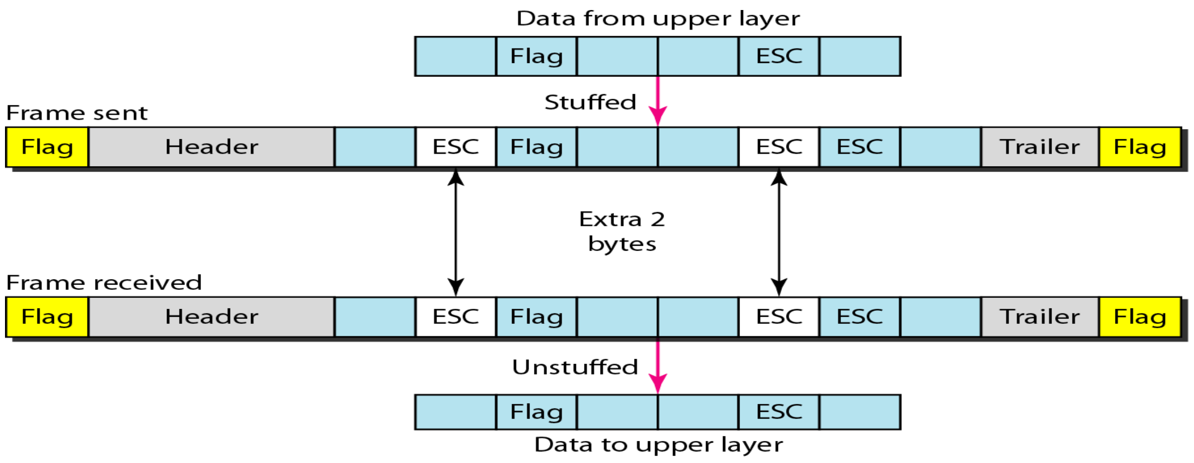


Fig. 3. 3 character-oriented protocol

- ***In byte stuffing (or character stuffing),** a special byte is added to the data section of the frame when there is a character with the same pattern as the flag. The data section is stuffed with an extra byte. This byte is usually called the **escape character (ESC)**, which has a predefined bit pattern.*
- *Whenever the receiver encounters the ESC character, it removes it from the data section and treats the next character as data, not a delimiting flag.*

❖ ***Byte stuffing is the process of adding 1 extra byte whenever there is a flag or escape character in the text.***

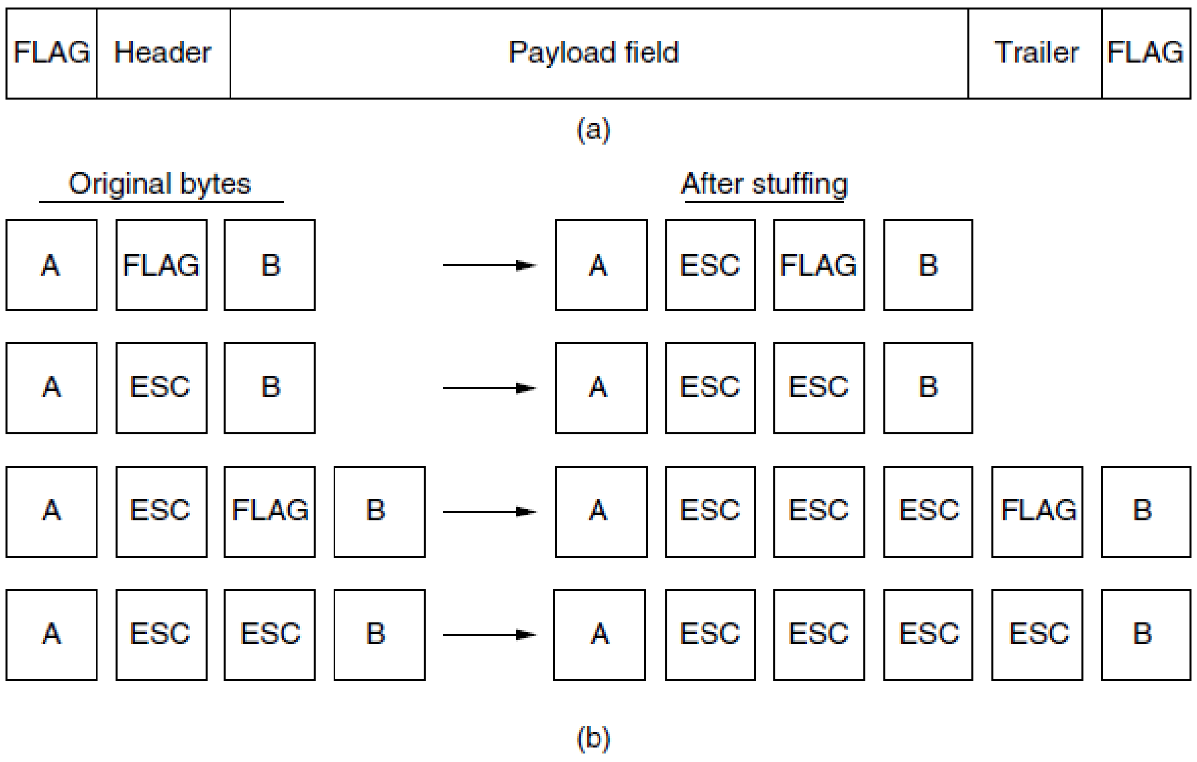


Fig: 3.4 character-oriented protocol

Starting and ending flags, with bit stuffing:

- *Bit stuffing is the process of adding one extra 0 whenever five consecutive 1s follow a 0 in the data, so that the receiver does not mistake the pattern 0111110 for a flag.*
- *In a **bit-oriented protocol**, the data section of a frame is a sequence of bits to be interpreted by the upper layer as text, graphic, audio, video, and so on.*
- *Most protocols use a special 8-bit pattern flag 01111110 as the delimiter to define the beginning and the end of the frame.*

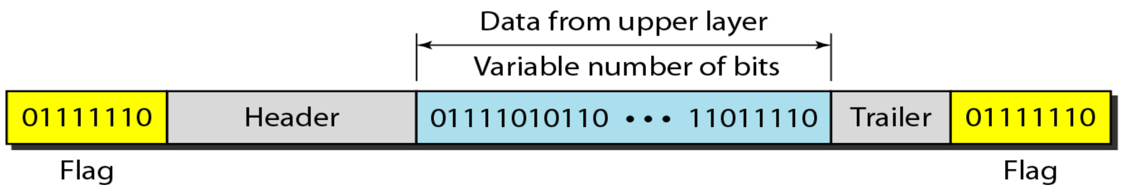


Fig: 3.5 A frame in a bit oriented protocol

- *This flag can create the same type of problem we saw in the byte-oriented protocols. That is, if the flag pattern appears in the data, we need to somehow inform the receiver that this is not the end of the frame. We do this by stuffing 1 single bit (instead of I byte) to prevent the pattern from looking like a flag. The strategy is called **bit stuffing.***

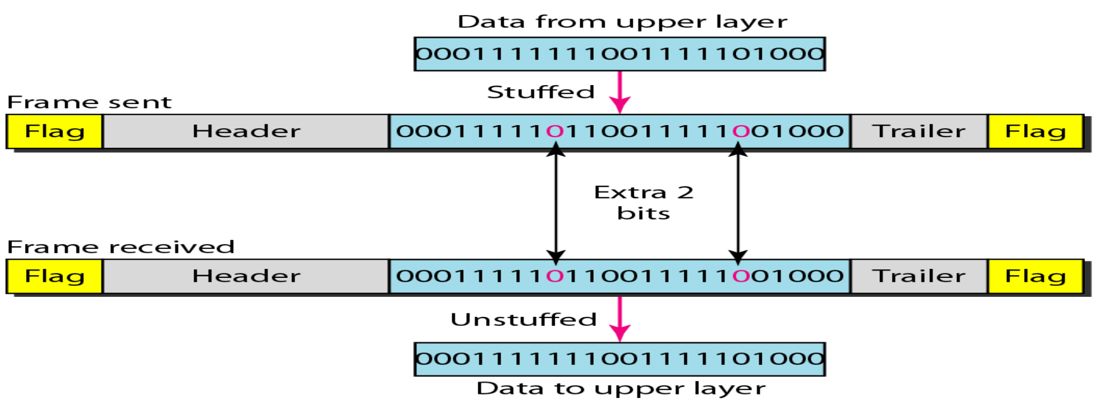


Fig: 3.6 Bit-Oriented Protocols

- In bit stuffing, if a 0 and five consecutive l bits are encountered, an extra 0 is added. This extra stuffed bit is eventually removed from the data by the receiver

Physical layer coding violations.

The last method of framing is only applicable to networks in which the encoding on the physical medium contains some redundancy. For example, some LANs encode 1 bit of data by using 2 physical bits. Normally, a 1 bit is a high-low pair and a 0 bit is a low-high pair. The scheme means that every data bit has a transition in the middle, making it easy for the receiver to locate the bit boundaries. The combinations high-high and low-low are not used for data but are used for delimiting frames in some protocols.

As a final note on framing, many data link protocols use combination of a character count with one of the other methods for extra safety. When a frame arrives, the count field is used to locate the end of the frame. Only if the appropriate delimiter is present at that position and the checksum is correct is the frame accepted as valid. Otherwise, the input stream is scanned for the next delimiter.

ERROR CONTROL:

- Error control includes 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.
- Error control in the data link layer is based on automatic repeat request (ARQ), which is the retransmission of data.
- Any time an error is detected in an exchange, specified frames are retransmitted. This process is called automatic repeat request (ARQ).

FLOW CONTROL:

- 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.
- 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.
- Since the rate of processing is often slower than the rate of transmission, 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 DETECTION AND CORRECTION:

- Data can be corrupted during transmission. Some applications require that errors be detected and corrected.
- Whenever bits flow from one point to another, they are subject to unpredictable changes because of interference. This interference can change the shape of the signal.

TYPES OF ERRORS:

There are two types. They are,

- **Single Bit Error**
- **Burst Error**

➢ Single Bit Error:

- It means that only one bit of a given data unit is changed from 1 to 0 or from 0 to 1. Single-bit errors are the least likely type of error in serial data transmission.

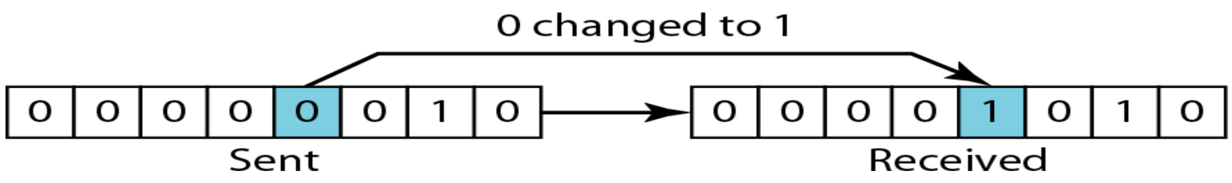


Fig: 3.7 Single Bit Error

➢ **Burst Error:**

- A burst error means that 2 or more bits in the data unit have changed.
- The number of bits affected depends on the data rate and duration of noise.
- The length of the burst is measured from the first corrupted bit to the last corrupted bit. Some bits in between may not have been corrupted.

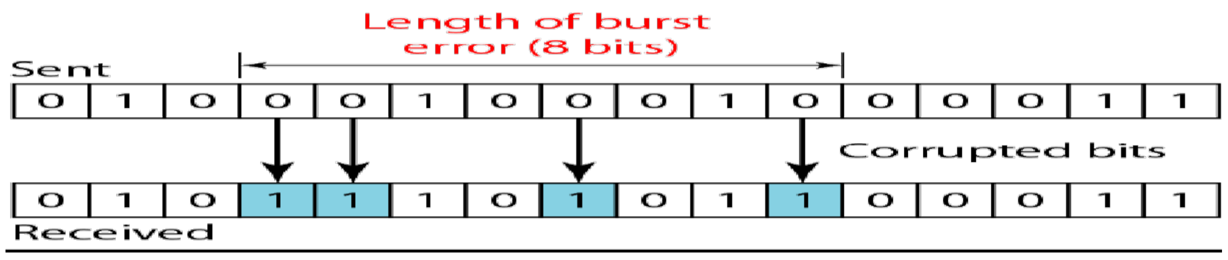


Fig: 3. 8 Burst error

ERROR DETECTION

For reliable communication errors must be detected and corrected. For error detection we are using many mechanisms.

REDUNDANCY

- Redundancy is a form of error detection mechanism is sending every data unit twice.
- The receiving device, the two units are compared and if they are same, it is assumed that no transmission errors have occurred. If any discrepancy would indicate an error, and an appropriate correction mechanism could be used.
- Redundancy is a character redundancy and message redundancy.
- When the data unit is a single character, it is called **Character redundancy.**
- When the data unit is a entire message, it is called **message redundancy.**
- But instead of repeating the entire data stream, a shorter group of bits may be appended to the end of each unit. This technique is called **redundancy** because extra bits are redundant to the information. They are discarded as soon as the accuracy of the transmission has been determined.

Types: Four types of redundancy checks are used in data communications. They are

- vertical redundancy check (VRC)
- longitudinal redundancy check (LRC)
- cyclic redundancy check (CRC)

- checksum

Detection versus Correction:

- The correction of errors is more difficult than the detection.
- In error detection, users are looking only to see if any error has occurred. A single-bit error is the same as a burst error.
- In error correction, user need to know the exact number of bits that are corrupted and their location in the message.
- The number of the errors and the size ofthe message are important factors.

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 is error-free.

Coding:

- Redundancy is achieved through various coding schemes.
- Coding schemes are divided into two broad categories:
  1. Block coding
  2. Convolution coding

Structure of encoder and decoder:

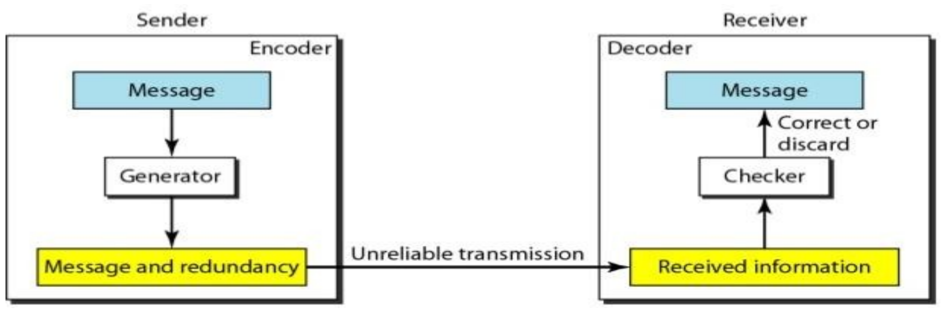


Fig: 3. 9 Structure of encoder and decoder

Modular Arithmetic:

- In modulo-N arithmetic, we use only the integers in the range 0 to $N - 1$, inclusive.

- ***Modulo-2 Arithmetic,***

Adding	*0+0=0*	*0+1=1*	*1+0=1*	*1+1=0*
Subtracting:	*0-0=0*	*0-1=1*	*1-0=1*	*1-1=0*

- *In this arithmetic we use the XOR (exclusive OR) operation for both addition and subtraction. The result of an XOR operation is 0 if two bits are the same; the result is I if two bits are different.*

BLOCK CODING:

- *In block coding, message is divided into blocks, each of size k bits, called as* ***datawords.***
- *Redundant bits(r) is add to each block to make the length n = k + r. The resulting n-bit blocks are called* ***codewords****.*
- *With k bits, we can create a combination of* ***2^kdatawords****; with n bits, we can create a combination of* ***2^ncodewords.***
- *Since* ***n >k****, the number of possible codewords is larger than the number of possible datawords.*
- *The block coding process is one-to-one; the same dataword is always encoded asthe same codeword.*

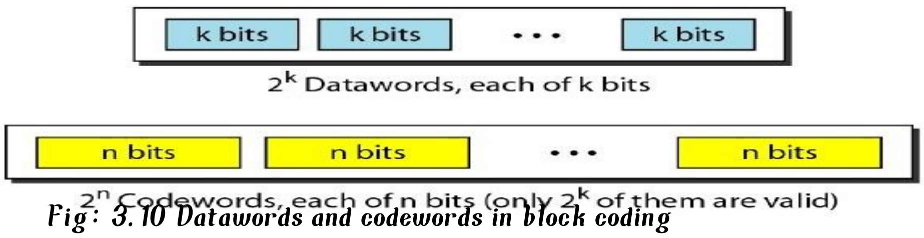

k bits
k bits
• • •
k bits
2^k Datawords, each of k bits
n bits
n bits
• • •
n bits
2^n Codewords, each of n bits (only 2^k of them are valid)

Fig: 3.10 Datawords and codewords in block coding

Error Detection:

Following steps are used for detecting errors in the block coding.

1. *The receiver has a list of valid codewords.*
2. *The original codeword has changed to an invalid one.*

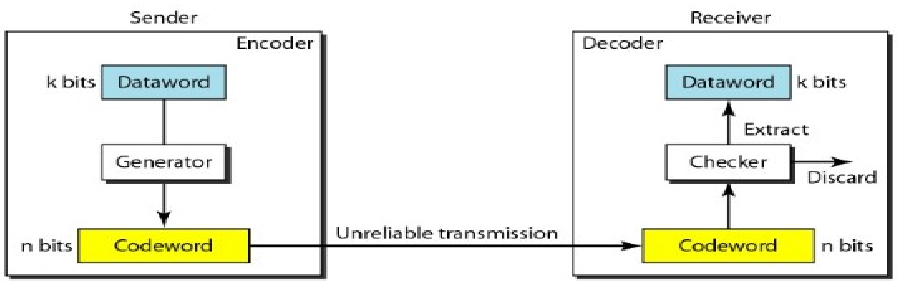
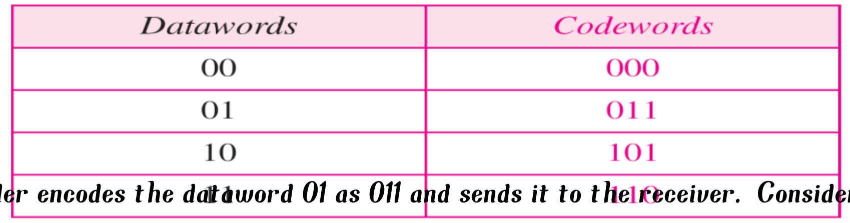


Fig: 3.11 Process of error detection in block coding

- *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.*
- *If the codeword is corrupted during transmission but the received word still matches a valid codeword, the error remains undetected.*
- *This type of coding can* ***detect only single errors. Two or more errors may remain undetected.***

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



Datawords	Codewords
00	000
01	011
10	101
11	110

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

1. *The receiver receives 011. It is a valid codeword. The receiver extracts the dataword 01 from it.*
2. *The codeword is corrupted during transmission, and 111 is received (the leftmost bit is corrupted). This is not a valid codeword and is discarded.*

3. The codeword is corrupted during transmission, and 000 is received (the right two bits are corrupted). This is a valid codeword.
4. The receiver incorrectly extracts the dataword 00.

Two corrupted bits have made the error undetectable.

Error Correction:

- Error correction is much more difficult than error detection.
- In error detection, the receiver needs to know only that the received codeword is invalid;
- In error correction the receiver needs to find the original codeword sent. More number of redundant bits are required for error correction than for error detection.

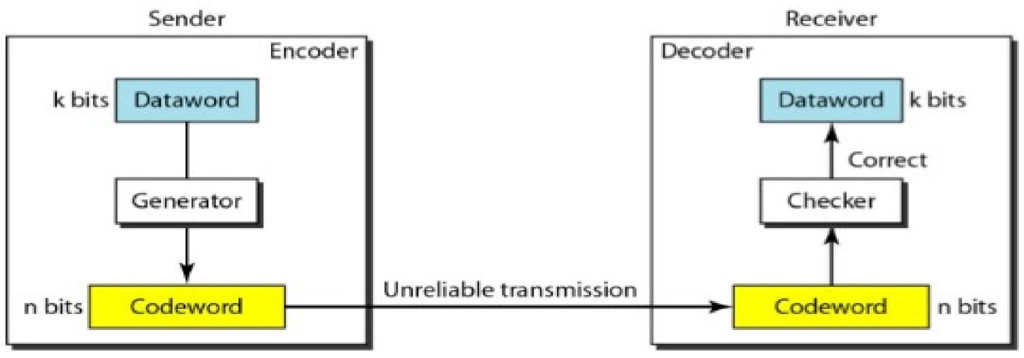


Fig: 3.12 Error Correction

Example: Assume the dataword is 01. The sender consults the table (or uses an algorithm) to create thecodeword 01011. The codeword is corrupted during transmission, and 01001 is received (error in the second bit from the right). First, the receiver finds that the received codeword is not in the table. This means an error has occurred. (Detection must come before correction.) The receiver, assuming that there is only 1 bit corrupted, uses the following strategy to guess the correct dataword.

Dataword	Codeword
00	00000
01	01011
10	10101
11	11110

1. Comparing the received codeword with the first codeword in the table (01001 versus 00000), the receiver decides that the first codeword is not the one that was sent because there are two different bits.
2. By the same reasoning, the original codeword cannot be the third or fourth one in the table.
3. The original codeword must be the second one in the table because this is the only one that differs from the received codeword by 1 bit. The receiver replaces 01001 with 01011 and consults the table to find the dataword 01.

Simple Parity-Check Code:

- The most familiar error-detecting code is the 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.
  Although some implementations specify an odd number of ls, we discuss the evencase.
- Most common, least complex
- Single bit is added to a block
- Two schemes:
  1. Even parity - Maintain even number of 1s
     - E. g. , 1011 → 1011**1**
  2. Odd parity - Maintain odd number of 1s
     - E. g. , 1011 → 1011**0**

Suppose the sender wants to send the word *world*. In ASCII the five characters are coded (with ***even parity***) as **1110111 1101111 1110010 1101100 1100100**

The following shows the actual bits sent

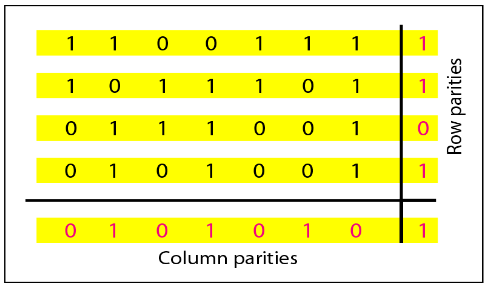
1110111**0** 1101111**0** 1110010**0** 1101100**0** 1100100**1**

Receiver receives this sequence of words:

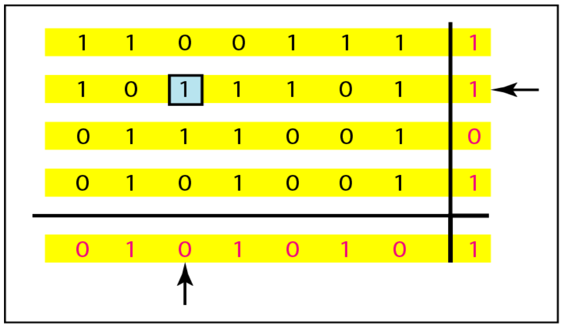
11111110 11011110 11101100 11011000 11001001

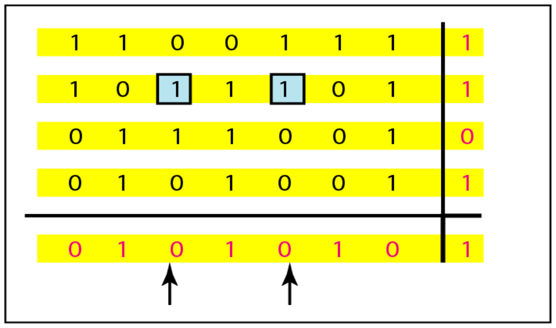
Which blocks are accepted? Which are rejected? 1 and 3 are rejected.

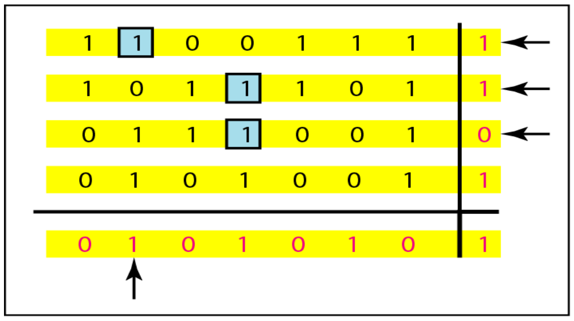
2D Parity Check



2D Parity Check: Performance


b. One error affects two parities


c. Two errors affect two parities


d. Three errors affect four parities

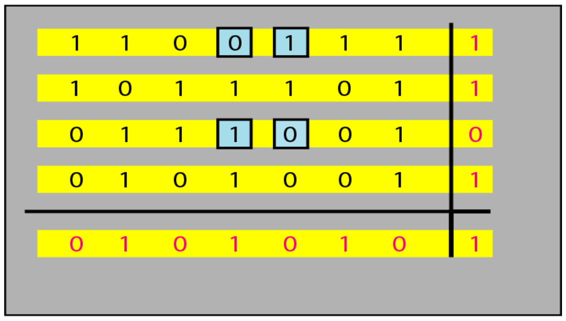

e. Four errors cannot be detected

Fig: 3.13 parity check codes

CHECKSUM

Idea

The concept of the checksum is not difficult.

- 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.
- **One's complement arithmetic:** 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).

The error detection method used by the higher layer protocols is called checksum. It consists of two parts. They are,

1. checksum generator
2. checksum checker

Checksum Generator:

In the sender, the checksum generator subdivides the data unit into equal segments of n bits. These segments are added with each other by using **one's complement arithmetic** in such a way that the total is also n bits long. That total is then complemented and appended to the end of the data unit.

Checksum Checker:

The receiver subdivides the data unit as above and adds all segments together and complements the result. If the extended data unit is intact, the total value found by adding the data segments and the checksum field should be zero. Otherwise the packet contains an error and the receiver rejects it.

EXAMPLE

At the sender

```
Data unit:10101001 00111001
               10101001
               00111001
     Sum       11100010
     Checksum  00011101
```

At the receiver

```
Received data: 10101001 00111001 00011101
               10101001
               00111001
               00011101
```

Sum	11111111
Complement	00000000

It means that the patter is ok.

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 0.
3. All words including the checksum are added using 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.

CYCLIC REDUNDANCY CHECK (CRC):

- CRC is based on binary division. In this a sequence of redundant bits, called CRC remainder is appended to the end of a data unit so that the resulting data unit becomes exactly divisible by a second predetermined binary number.
- At its destination, the incoming data unit is divided by the same number. If at this step there is no reminder, the data unit is assumed to be intact and therefore accepted. A remainder indicates that the data unit has been changed in transit and therefore must be rejected.
- Here, the remainder is the CRC. It must have exactly one less bit than the divisor, and appending it to the end of the data string must make the resulting bit sequence exactly divisible by the divisor.

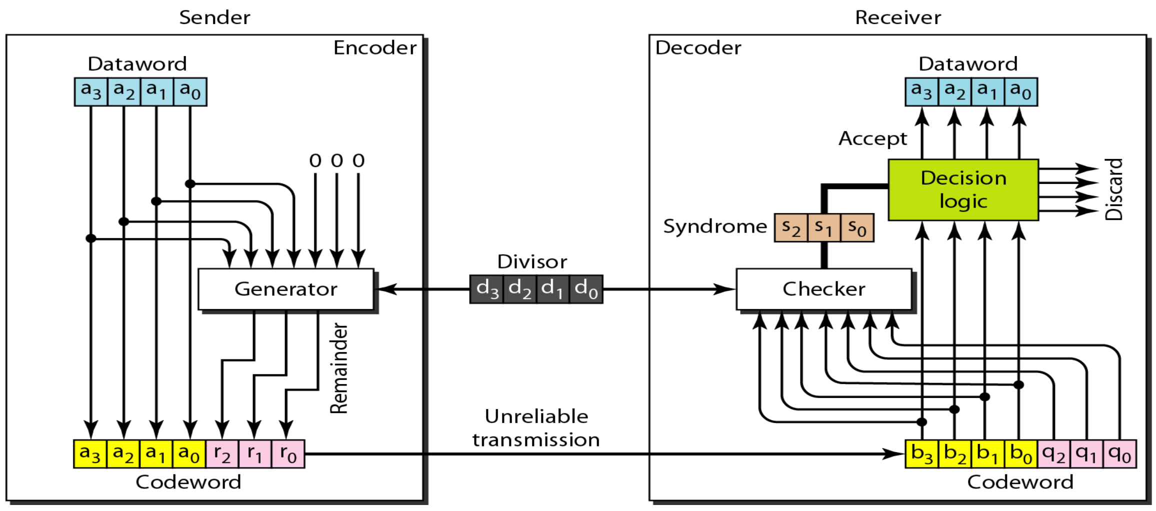


Fig: 3.14 CYCLIC REDUNDANCY CHECK (CRC)

- First, a string of n-1 0s is appended to the data unit. The number of 0s is one less than the number of bits in the divisor which is n bits. Then the newly elongated data unit is divided by the divisor using a process called binary division. The remainder is CRC. The CRC is replaces the appended 0s at the end of the data unit
- The data unit arrives at the receiver first, followed by the CRC. The receiver treats whole string as the data unit and divides it by the same divisor that was used to find the CRC remainder. If the remainder is 0 then the data unit is error free. Otherwise it having some error and it must be discarded.
- **CRC Generator**

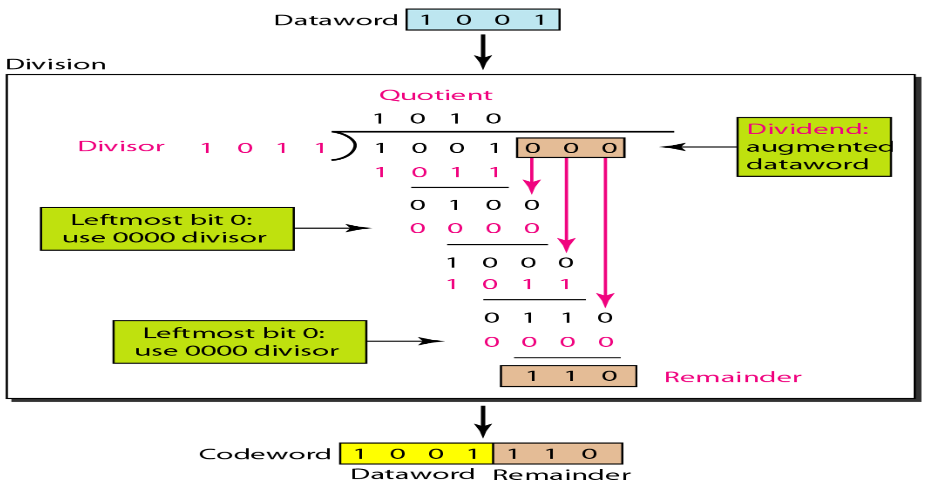


Fig: 3.15 CRC Generator

- ***Checking CRC***

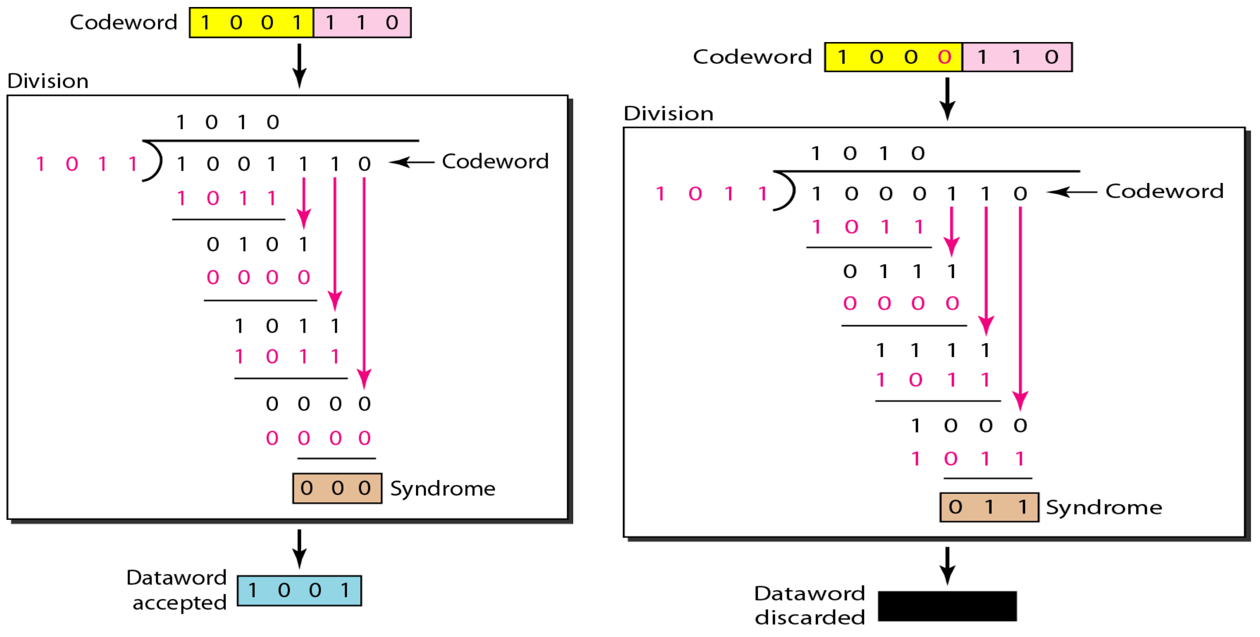


Fig: 3.16 Checking CRC

Polynomial codes

- *A pattern of 0s and 1s can be represented as a polynomial with coefficient of 0 and 1.*
- *Here, the power of each term shows the position of the bit and the coefficient shows the values of thebit.*
- *For example, if binary pattern is 100101, its corresponding polynomial representation is x^5 + x^2 + 1. Figure shows the polynomial where all the terms with zero coefficient are removed and x J is replaced by x and XO by 1.*

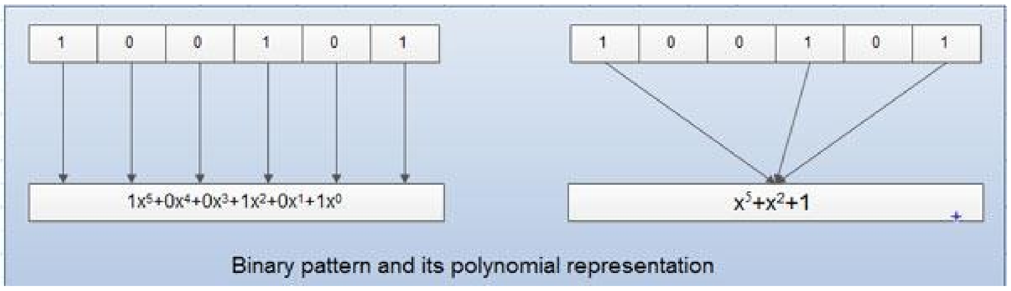


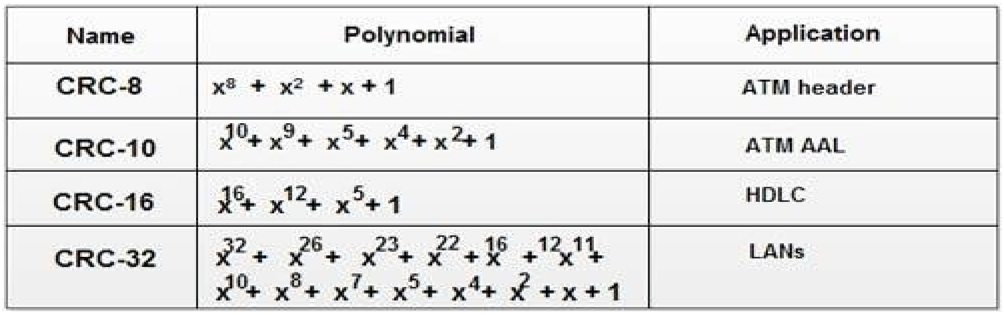
Fig:3.17 Polynomial representation

- *The benefits of using polynomial codes is that it produces short codes. For example here a 6-bit pattern is replaced by 3 terms.*

- *In polynomial codes, the degree is 1 less than the number of bits in the binary pattern. The degree of polynomial is the highest power in polynomial. For example as shown in fig degree of polynomial x^5+x^2+1 are 5. The bit pattern in this case is 6.*
- *The polynomial generator should have following* ***properties****:*

  1. *It should have at least two terms.*
  2. *The coefficient of the term x^0 should be 1.*
  3. *It should not be divisible by x.*
  4. *It should be divisible by x+ 1.*

- *There are several different standard polynomials used by popular protocols for CRC generation. These are:*



Name	Polynomial	Application
CRC-8	$x^8 + x^2 + x + 1$	ATM header
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$	ATM AAL
CRC-16	$x^{16} + x^{12} + x^5 + 1$	HDLC
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	LANs

<u>Error-Correcting Codes:</u>

We will examine four different error-correcting codes:

1. *Hamming codes.*
2. *Binary convolutional codes.*
3. *Reed-Solomon codes.*
4. *Low-Density Parity Check codes.*

All of these codes add redundancy to the information that is sent. A frame consists of m data (i. e., message) bits and r redundant (i. e. check) bits. In a ***block code****, the r check bits are computed solely as a function of the m data bits with which they are associated, as though the m bits were looked up in a large table to find their corresponding r check bits.*

In a **systematic code**, the *m* data bits are sent directly, along with the check bits, rather than being encoded themselves before they are sent. In a **linear code**, the *r* check bits are computed as a linear function of the *m* data bits.

Let the total length of a block be *n* (i. e., $n = m + r$). We will describe this as an (n, m) code. An *n*-bit unit containing data and check bits is referred to as an *n*bit **codeword**. The **code rate**, or simply rate, is the fraction of the codeword that carries information that is not redundant, or m/n.

The number of bit positions in which two codewords differ is called the **Hamming distance** (Hamming, 1950). Its significance is that if two codewords are a Hamming distance *d* apart, it will require *d* single-bit errors to convert one into the other.

Hamming codes:

Hamming code is a set of error-correction codes that can be used to **detect and correct the errors** that can occur when the data is moved or stored from the sender to the receiver. It is **technique developed by R. W. Hamming for error correction.**

Redundant bits –

Redundant bits are extra binary bits that are generated and added to the information-carrying bits of data transfer to ensure that no bits were lost during the data transfer. The number of redundant bits can be calculated using the following formula:

$2^r \geq m + r + 1$

where, r = redundant bit, m = data bit

Suppose the number of data bits is 7, then the number of redundant bits can be calculated using:

$= 2^4 \geq 7 + 4 + 1$

Thus, the number of redundant bits= 4

Parity bits –

A parity bit is a bit appended to a data of binary bits to ensure that the total number of 1's in the data are even or odd. Parity bits are used for error detection. There are two types of parity bits:

1. **Even parity bit:**

   In the case of even parity, for a given set of bits, the number of 1' s are counted. If that count is odd, the parity bit value is set to 1, making the total count of occurrences of 1' s an even number. If the total number of 1' s in a given set of bits is already even, the parity bit' s value is 0.

2. **Odd Parity bit –**

   In the case of odd parity, for a given set of bits, the number of 1' s are counted. If that count is even, the parity bit value is set to 1, making the total count of occurrences of 1' s an odd number. If the total number of 1' s in a given set of bits is already odd, the parity bit' s value is 0.

General Algorithm of Hamming code –

The Hamming Code is simply the use of extra parity bits to allow the identification of an error.

1. Write the bit positions starting from 1 in binary form (1, 10, 11, 100, etc).
2. All the bit positions that are a power of 2 are marked as parity bits (1, 2, 4, 8, etc).
3. All the other bit positions are marked as data bits.
4. Each data bit is included in a unique set of parity bits, as determined its bit position in binary form.

   a. Parity bit 1 covers all the bits positions whose binary representation includes a 1 in the least significant
   position (1, 3, 5, 7, 9, 11, etc).

   b. Parity bit 2 covers all the bits positions whose binary representation includes a 1 in the second position from
   the least significant bit (2, 3, 6, 7, 10, 11, etc).

   c. Parity bit 4 covers all the bits positions whose binary representation includes a 1 in the third position from the least significant bit (4–7, 12–15, 20–23, etc).

d. Parity bit 8 covers all the bits positions whose binary representation includes a 1 in the fourth position from
the least significant bit bits (8–15, 24–31, 40–47, etc).

e. In general each parity bit covers all bits where the bitwise AND of the parity position and the bit position is
non-zero.

5. Since we check for even parity set a parity bit to 1 if the total number of ones in the positions it checks is
odd.

6. Set a parity bit to 0 if the total number of ones in the positions it checks is even.

Determining the position of redundant bits –

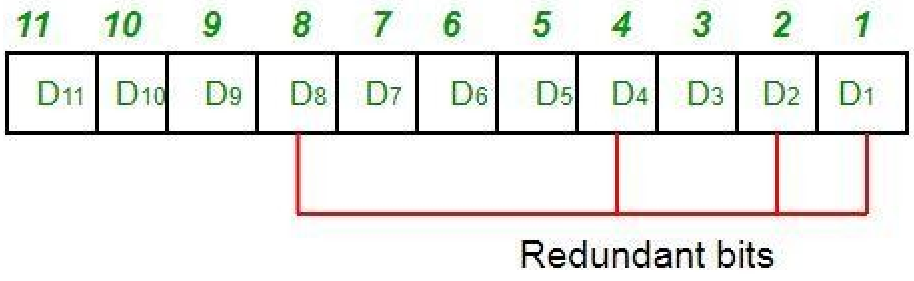
These redundancy bits are placed at the positions which correspond to the power of 2. As in the above example:

1. The number of data bits = 7

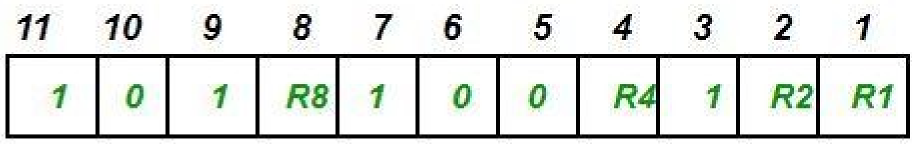
2. The number of redundant bits = 4

3. The total number of bits = 11

4. The redundant bits are placed at positions corresponding to power of 2- 1, 2, 4, and 8



Suppose the data to be transmitted is 1011001, the bits will be placed as follows:

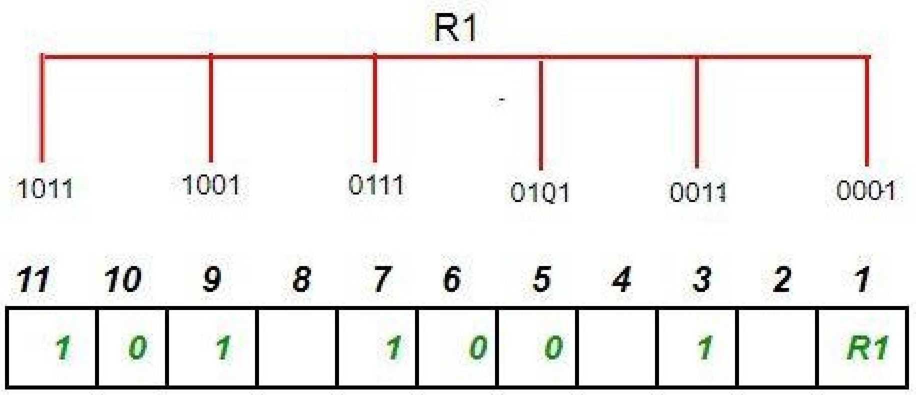


11	10	9	8	7	6	5	4	3	2	1
1	0	1	R8	1	0	0	R4	1	R2	R1

Determining the Parity bits –

1. R1 bit is calculated using parity check at all the bits positions whose binary representation includes a 1 in the least significant position.

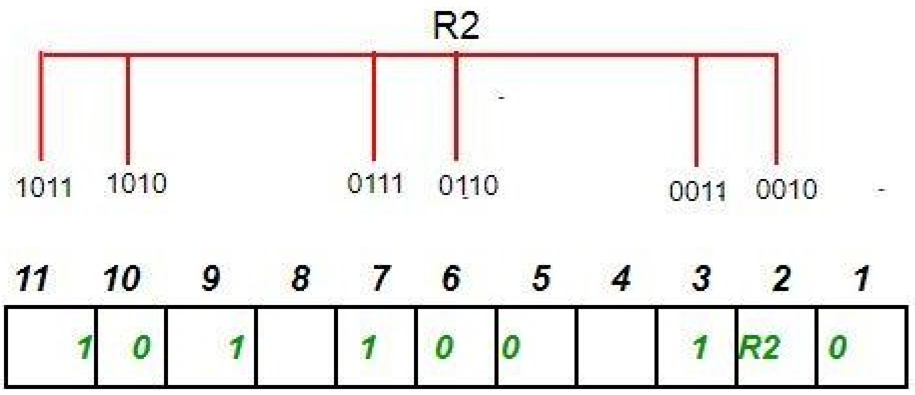
R1: bits 1, 3, 5, 7, 9, 11



To find the redundant bit R1, we check for even parity. Since the total number of 1's in all the bit positions corresponding to R1 is an even number the value of R1 (parity bit's value) = 0

2. R2 bit is calculated using parity check at all the bits positions whose binary representation includes a 1 in the second position from the least significant bit.

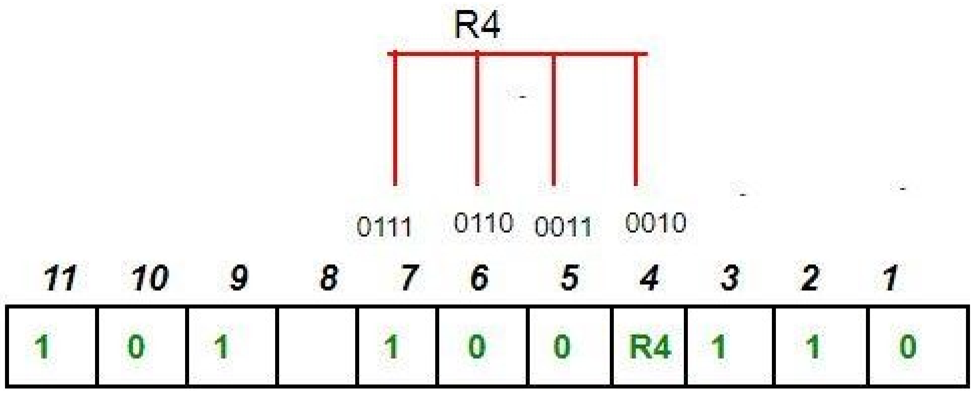
R2: bits 2, 3, 6, 7, 10, 11



To find the redundant bit R2, we check for even parity. Since the total number of 1's in all the bit positions corresponding to R2 is an odd number the value of R2(parity bit's value)=1

3. *R4 bit is calculated using parity check at all the bits positions whose binary representation includes a 1 in the third position from the least significant bit.*

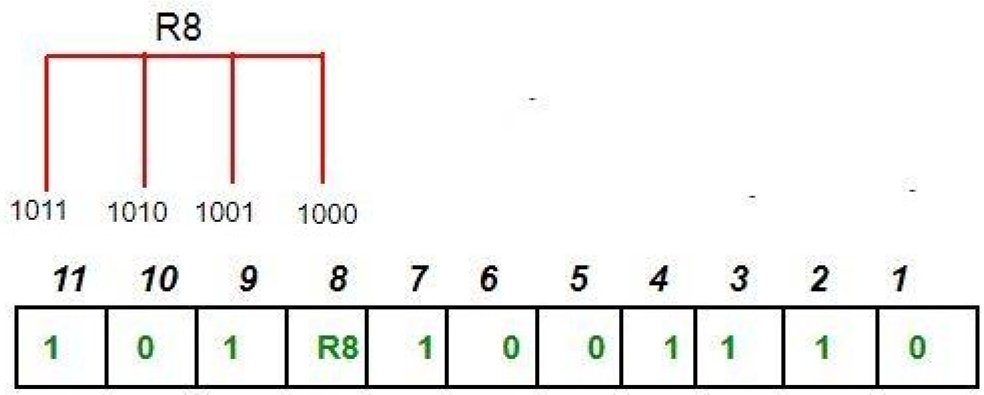
R4: bits 4, 5, 6, 7



To find the redundant bit R4, we check for even parity. Since the total number of 1's in all the bit positions corresponding to R4 is an odd number the value of R4(parity bit's value) = 1

4. *R8 bit is calculated using parity check at all the bits positions whose binary representation includes a 1 in the fourth position from the least significant bit.*

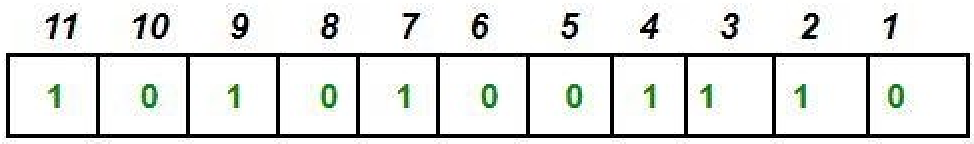
R8: bit 8,9,10,11



11	10	9	8	7	6	5	4	3	2	1
1	0	1	R8	1	0	0	1	1	1	0

To find the redundant bit R8, we check for even parity. Since the total number of 1's in all the bit positions corresponding to R8 is an even number the value of R8(parity bit's value)=0.

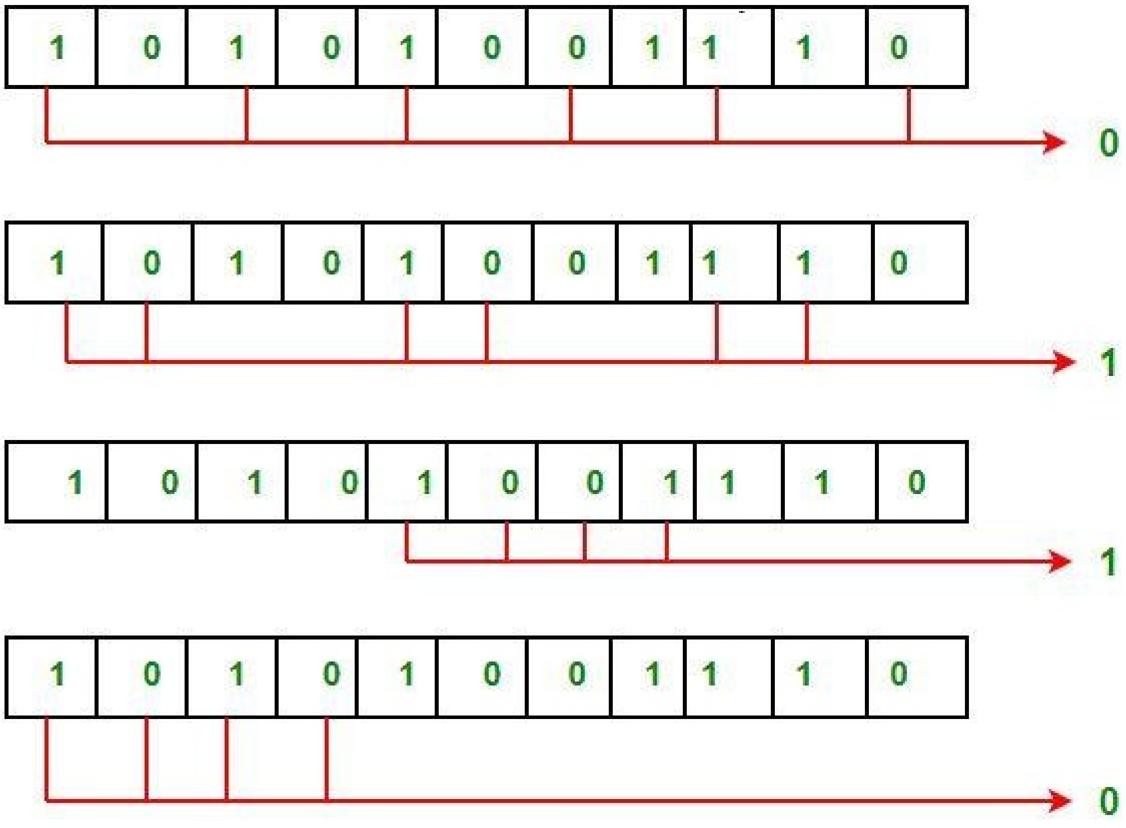
Thus, the data transferred is:



11	10	9	8	7	6	5	4	3	2	1
1	0	1	0	1	0	0	1	1	1	0

Error detection and correction –

Suppose in the above example the 6th bit is changed from 0 to 1 during data transmission, then it gives new parity values in the binary number:

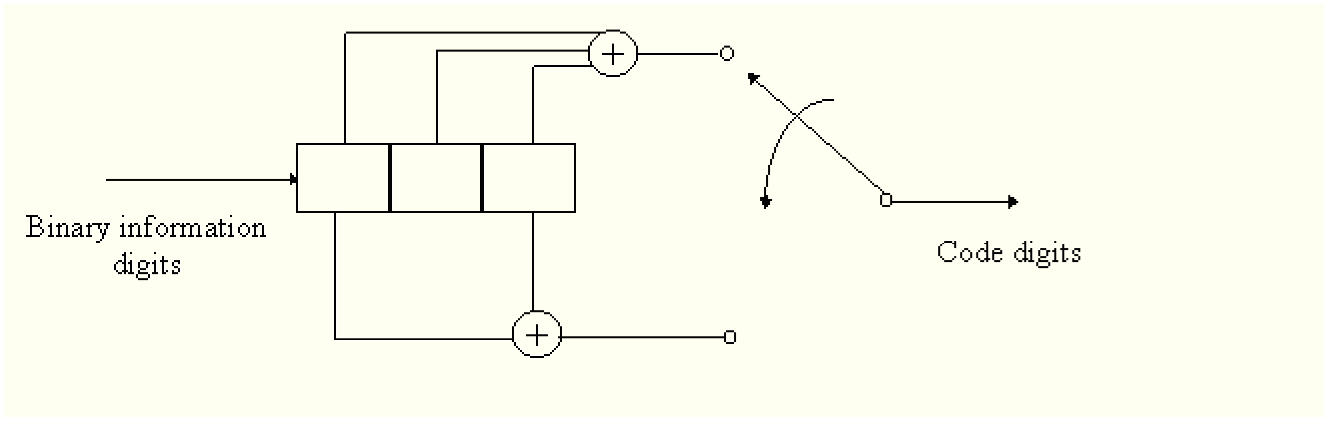


The bits give the binary number as 0110 whose decimal representation is 6. Thus, the bit 6 contains an error. To correct the error the 6th bit is changed from 1 to 0.

Binary convolutional code:

The second code we will look at is a ***convolutional code****. This code is the only one we will cover that is not a block code. In a convolutional code, an encoder processes a sequence of input bits and generates a sequence of output bits. There is no natural message size or encoding boundary as in a block code. The output depends on the current and previous input bits. That is, the encoder has memory. The number of previous bits on which the output depends is called the* ***constraint length*** *of the code. Convolutional codes are specified in terms of their rate and constraint length.*

A simple rate ½ convolutional code ***encoder*** *is shown below.*



The rectangular box represents one element of a ***serial shift register****. The contents of the shift registers is shifted from left to right. The circle with a plus sign inside it represents XOR operation. Notice how the* ***code digits*** *which are output by the encoder are multiplexed into a serial stream of binary digits. For every binary digit that enters the encoder, two code digits are output. Hence code rate = 1/2.*

In general, the code rate $R_{code} = k / n$

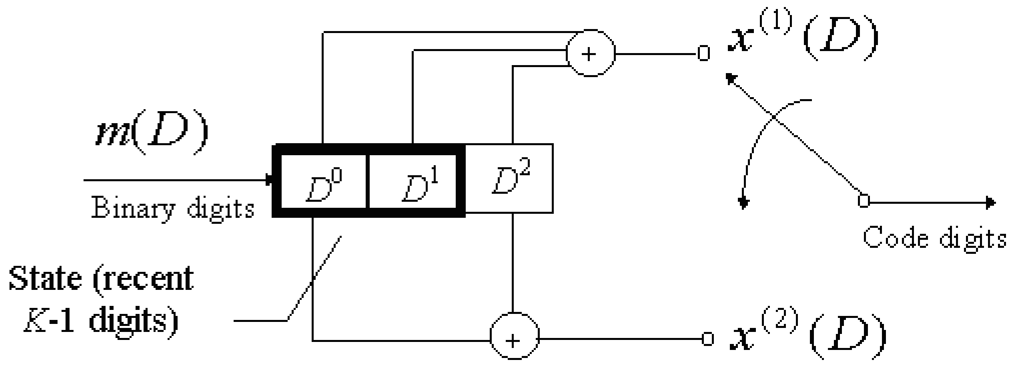
The constraint length K of a convolutional code is defined as the number of shifts over which a single message bit can influence the encoder output.

HOW TO DETERMINE THE OUTPUT CODEWORD?

There are essentially two ways in which the output codeword may be determined. We will take a close look at both methods in this lecture.

State Diagram Approach

The state of a rate 1/2 encoder is defined by the most (K-1) message bits moved into the encoder as illustrated in the diagram below.



- The corresponding state diagram for the rate 1/2, $K = 3$ convolutional code is shown below. Notice that there are four states [00] , [01], [10], [11], corresponding to the $(K-1)=2$ binary digit tuple.

- We may assume the encoder starts in the all-zero state [00]

STATE TABLE

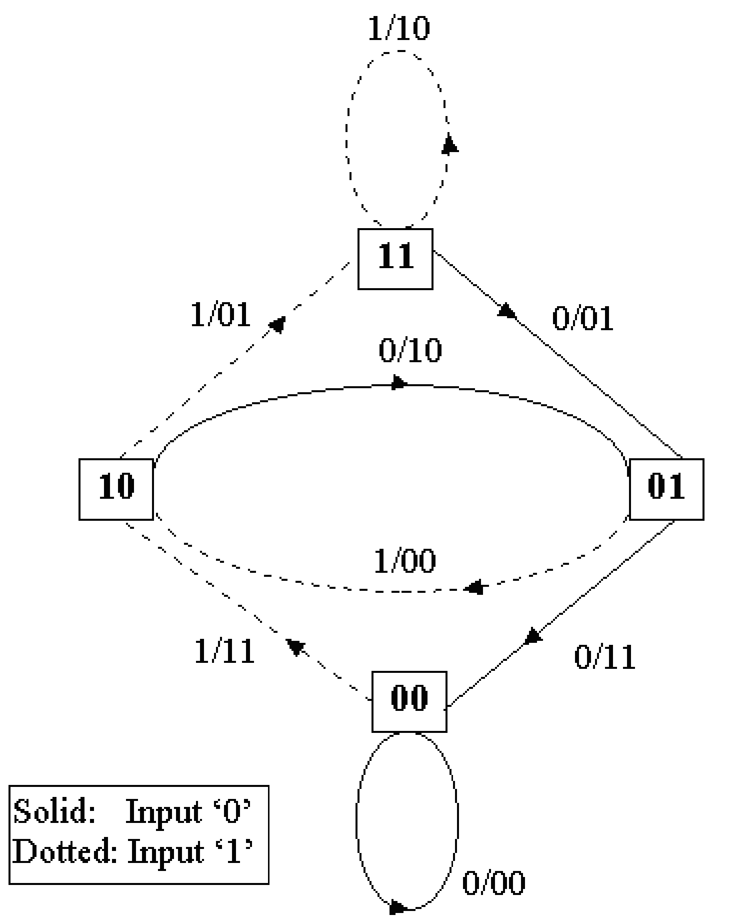
The easiest way to determine the state diagram is to first determine the state table as shown below.

INPUT DIGIT	STATE	FINAL STATE	CODEWORD
0	00	00	00
1	00	10	11
0	01	00	11
1	01	10	00
0	10	01	10
1	10	11	01
0	11	01	01
1	11	11	10

- Notice how all the combinations of the initial state and input digit are used to first determine the final state, and then the output codeword.

- Now we use this state table to draw the state diagram as shown below.

STATE DIAGRAM



KEY

1/01 This means for example, that the input binary digit to the encoder was 1 and the corresponding codeword output is 01.

$\boxed{01}$ This represents the state of the encoder. In this example, the state is 01

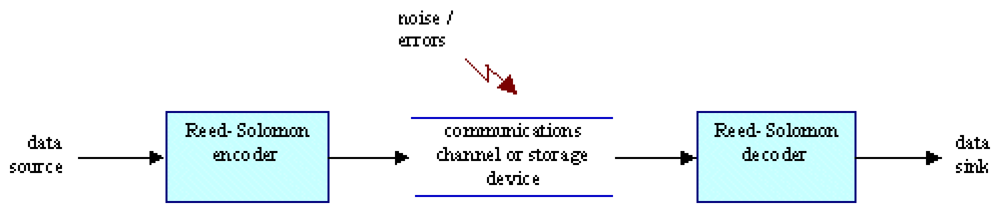
Reed-Solomon code:

Reed-Solomon codes are block-based error correcting codes with a wide range of applications in digital communications and storage. Reed-Solomon codes are used to correct errors in many systems including:

- Storage devices (including tape, Compact Disk, DVD, barcodes, etc)
- Wireless or mobile communications (including cellular telephones, microwave links, etc)

- Satellite communications
- Digital television / DVB
- High-speed modems such as ADSL, xDSL, etc.

A typical system is shown here:



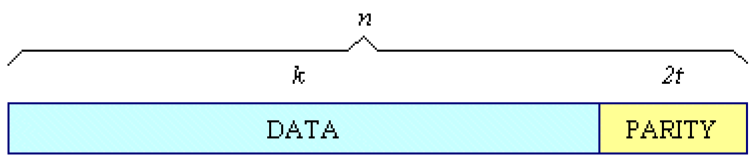
The Reed-Solomon encoder takes a block of digital data and adds extra "redundant" bits. Errors occur during transmission or storage for a number of reasons (for example noise or interference, scratches on a CD, etc). The Reed-Solomon decoder processes each block and attempts to correct errors and recover the original data. The number and type of errors that can be corrected depends on the characteristics of the Reed-Solomon code.

2. Properties of Reed-Solomon codes

Reed Solomon codes are a subset of BCH codes and are linear block codes. A Reed-Solomon code is specified as RS(n, k) with s-bit symbols.

This means that the encoder takes k data symbols of s bits each and adds parity symbols to make an n symbol codeword. There are $n-k$ parity symbols of s bits each. A Reed-Solomon decoder can correct up to t symbols that contain errors in a codeword, where $2t = n-k$.

The following diagram shows a typical Reed-Solomon codeword (this is known as a Systematic code because the data is left unchanged and the parity symbols are appended):



Example: A popular Reed-Solomon code is RS(255, 223) with 8-bit symbols. Each codeword contains 255 code word bytes, of which 223 bytes are data and 32 bytes are parity. For this code:

$n = 255,\ k = 223,\ s = 8$

$2t = 32,\ t = 16$

The decoder can correct any 16 symbol errors in the code word: i. e. errors in up to 16 bytes anywhere in the codeword can be automatically corrected.

Given a symbol size s, the maximum codeword length (n) for a Reed-Solomon code is $\mathbf{n = 2^s - 1}$

For example, the maximum length of a code with 8-bit symbols (s=8) is 255 bytes.

Reed-Solomon codes may be shortened by (conceptually) making a number of data symbols zero at the encoder, not transmitting them, and then re-inserting them at the decoder.

Example: The (255, 223) code described above can be shortened to (200, 168). The encoder takes a block of 168 data bytes, (conceptually) adds 55 zero bytes, creates a (255, 223) codeword and transmits only the 168 data bytes and 32 parity bytes.

The amount of processing "power" required to encode and decode Reed-Solomon codes is related to the number of parity symbols per codeword. A large value of t means that a large number of errors can be corrected but requires more computational power than a small value of t.

Symbol Errors

One symbol error occurs when 1 bit in a symbol is wrong or when all the bits in a symbol are wrong.

Example: RS(255, 223) can correct 16 symbol errors. In the worst case, 16 bit errors may occur, each in a separate symbol (byte) so that the decoder corrects 16 bit errors. In the best case, 16 complete byte errors occur so that the decoder corrects 16 x 8 bit errors.

Reed-Solomon codes are particularly well suited to correcting burst errors (where a series of bits in the codeword are received in error).

Decoding

Reed-Solomon algebraic decoding procedures can correct errors and erasures. An erasure occurs when the position of an erred symbol is known. A decoder can correct up to *t* errors or up to *2t* erasures. Erasure information can often be supplied by the demodulator in a digital communication system, i. e. the demodulator "flags" received symbols that are likely to contain errors.

When a codeword is decoded, there are three possible outcomes:

1. If $2s + r < 2t$ (s errors, r erasures) then the original transmitted code word will always be recovered,

OTHERWISE

2. The decoder will detect that it cannot recover the original code word and indicate this fact.

OR

3. The decoder will mis-decode and recover an incorrect code word without any indication.

The probability of each of the three possibilities depends on the particular Reed-Solomon code and on the number and distribution of errors.

Coding Gain

The advantage of using Reed-Solomon codes is that the probability of an error remaining in the decoded data is (usually) much lower than the probability of an error if Reed-Solomon is not used. This is often described as **coding gain.**

Example: A digital communication system is designed to operate at a Bit Error Ratio (BER) of 10^{-9}, i. e. no more than 1 in 10^{9} bits are received in error. This can be achieved by boosting the power of the transmitter **or** by adding Reed-Solomon (or another type of Forward Error Correction). Reed-Solomon allows the system to achieve this target BER with a lower transmitter output power. The power "saving" given by Reed-Solomon (in decibels) is the **coding gain.**

Low-Density Parity Check

The final error-correcting code we will cover is the **LDPC** (**Low-Density Parity Check**) code. LDPC codes are linear block codes that were invented by Robert Gallagher in his doctoral thesis (Gallagher, 1962). Like most theses, they were promptly forgotten, only to be reinvented in 1995 when advances in computing power had made them practical. In an LDPC code, each output bit is formed from only a fraction of the input bits. This leads to a matrix representation of the code that has a low density of 1s, hence the name for the code. The received codewords are decoded with an approximation algorithm that iteratively improves on a best fit of the received data to a legal codeword. This corrects errors.

ELEMENTARY DATA LINK LAYER PROTOCOLS

When data-frame is transmitted, there is a probability that data-frame may be lost in the transit or it is received corrupted. In both cases, the receiver does not receive the correct data-frame and sender does not know anything about any loss. In such case, both sender and receiver are equipped with some protocols which helps them to detect transit errors such as loss of data- frame. Hence, either the sender retransmits the data-frame or the receiver may request to resend the previous data-frame.

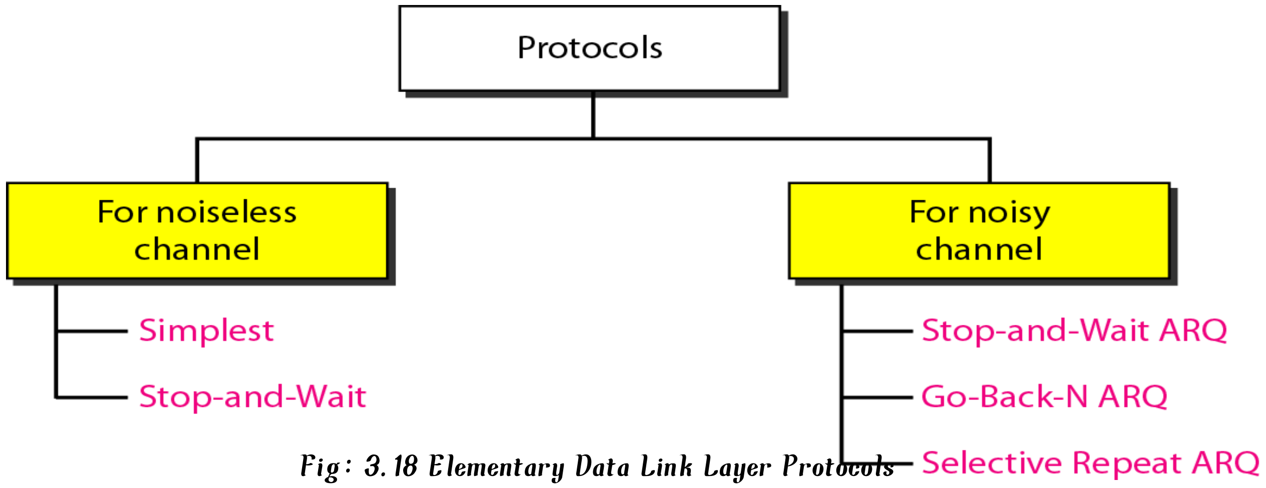


Fig: 3.18 Elementary Data Link Layer Protocols

- **Positive ACK** – When the receiver receives a correct frame, it should acknowledge it.
- **Negative ACK** – When the receiver receives a damaged frame or a duplicate frame, it sends a NACK back to the sender and the sender must retransmit the correct frame.

- ***Retransmission:*** *The sender maintains a clock and sets a timeout period. If an acknowledgement of a data-frame previously transmitted does not arrive before the timeout the sender retransmits the frame, thinking that the frame or its acknowledgement is lost in transit.*
- ***Piggybacking:*** *In two way communication, whenever a data frame is received, the receiver waits and does not send the control frame (acknowledgement) back to the sender immediately. The receiver waits until its network layer passes in the next data packet. The delayed acknowledgement is then attached to this outgoing data frame. His technique of temporarily delaying the acknowledgement so that it can be hooked with next outgoing data frame is known as* ***piggybacking.***

NOISELESS CHANNELS

Simplest Protocol:

- *Simplest Protocol is one that has no flow or error control and it is a unidirectional protocol in which data frames are traveling in only one direction-from the sender to receiver.*
- *We assume that the receiver can immediately handle any frame it receives with a processing time that is small enough to be negligible. The data link layer of the receiver immediately removes the header from the frame and hands the data packet to its network layer, which can also accept the packet immediately.*
- *No sequence numbers or acknowledgements are used here.*

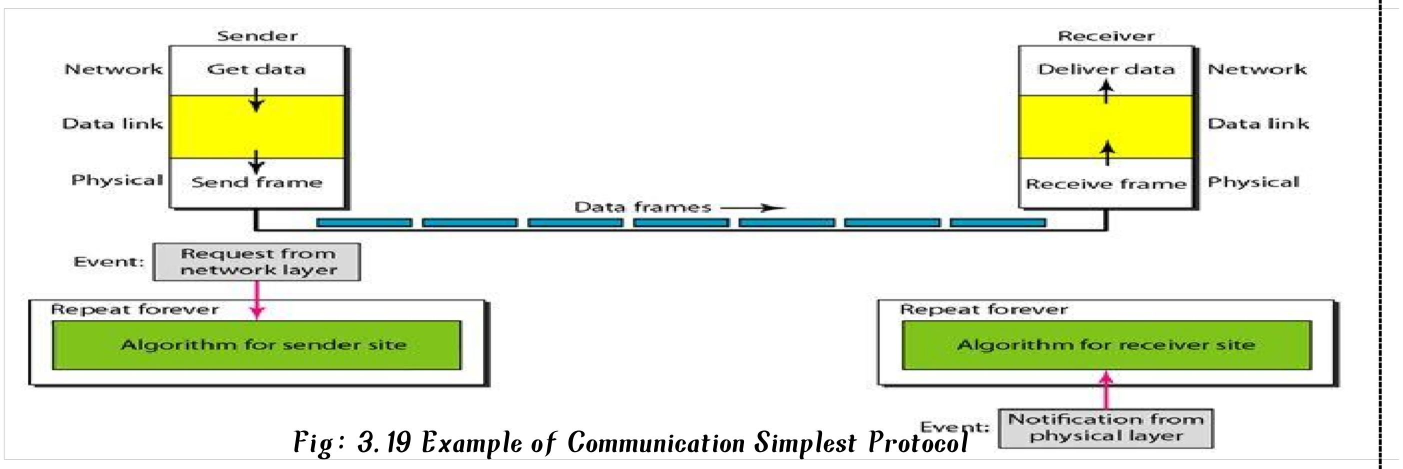


Fig: 3.19 Example of Communication Simplest Protocol

- *The following figure shows an example of communication using this protocol. It is very simple. The sender sends a sequence of frames without even thinking about the receiver. To send three frames, three events occur at the sender site and three events at the receiver site.*
- *Data are transmitted in one direction only*
- *The transmitting (Tx) and receiving (Rx) hosts are always ready*
- *Processing time can be ignored*
- *Infinite buffer space is available*
- *No errors occur; i. e. no damaged frames and no lost frames (perfect channel)*

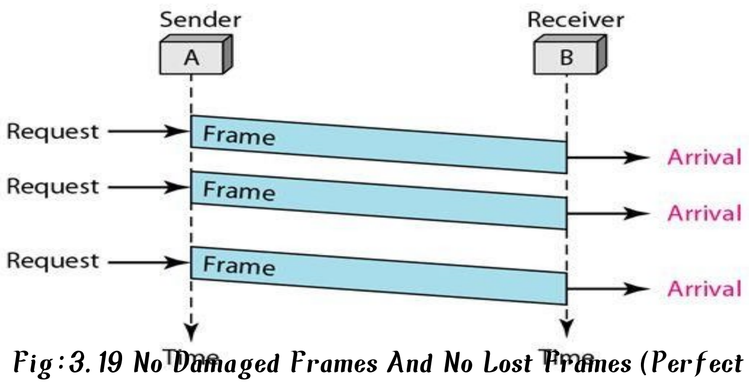


Fig:3.19 No Damaged Frames And No Lost Frames (Perfect Channel)

"Simplest": Pseudo Code

- *Sender*

```
while(true)                         // Repeat forever
{
  WaitForEvent();                   // Sleep until an event occurs
   if(Event(RequestToSend))         //There is a packet to send
   {
       GetData();
       MakeFrame();
       SendFrame();                 //Send the frame
   }
}
```

- *Receiver*

```
while(true)                         // Repeat forever
{
  WaitForEvent();                   // Sleep until an event occurs
  if(Event(ArrivalNotification))    //Data frame arrived
  {
     ReceiveFrame();
     ExtractData();
     DeliverData();                 //Deliver data to network layer
  }
}
```

Stop-and-Wait Protocol

- *If data frames arrive at the receiver site faster than they can be processed, the frames must be stored until their use. Normally, the receiver does not have enough storage space, especially if it is receiving data from many sources. This may result in either the discarding of frames or denial of service. To prevent the receiver from becoming overwhelmed with frames, we somehow need to tell the sender to slow down. There must be feedback from the receiver to the sender.*
- *The protocol now is called the Stop-and-Wait Protocol because the sender sends one frame, stops until it receives confirmation from the receiver (okay to go ahead), and then sends the next frame.*
- *It is a unidirectional communication for data frames, but auxiliary ACK frames (simple tokens of acknowledgment) travel from the other direction.*
- *Receiver has limited buffer, Requires flow control.*

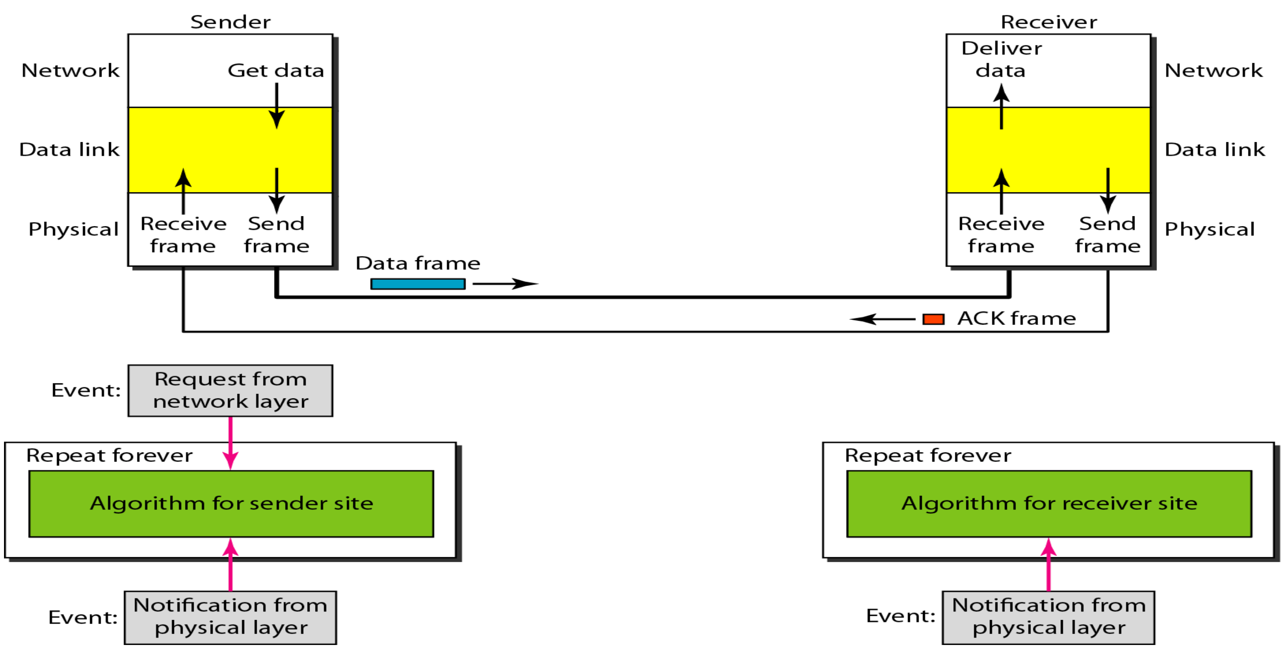
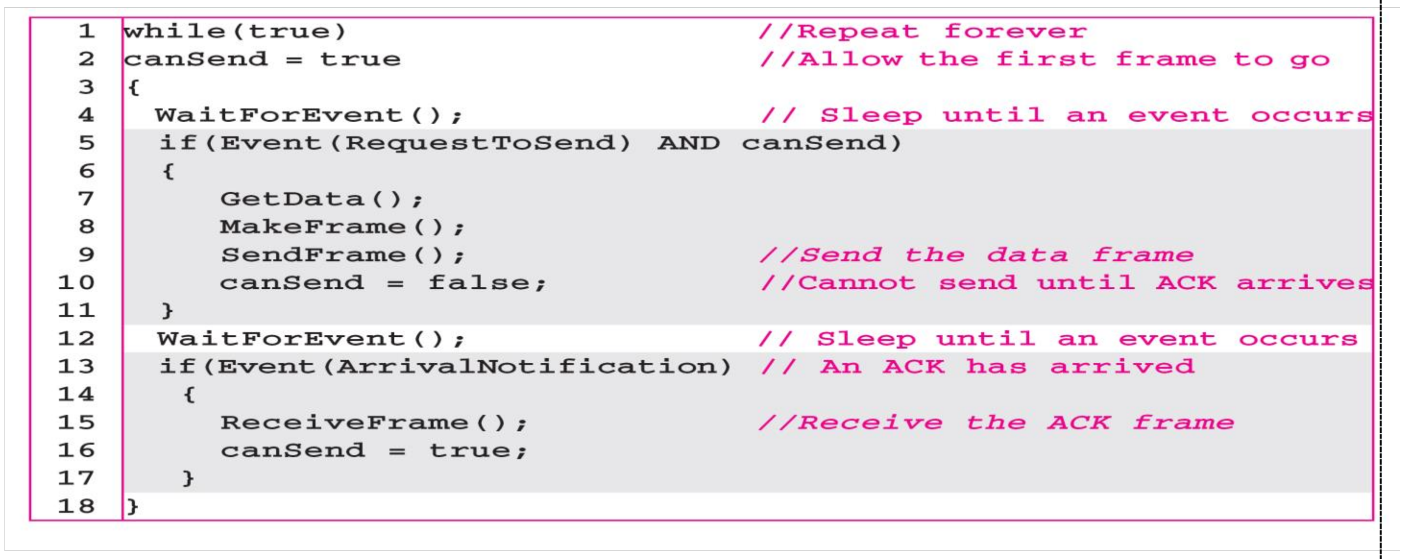


Fig: 3. 20 Stop-and-Wait Protocol

Stop-and-Wait: Pseudo Code

- ***Sender side***



```
while(true)                              //Repeat forever
canSend = true                           //Allow the first frame to go
{
  WaitForEvent();                        // Sleep until an event occurs
  if(Event(RequestToSend) AND canSend)
  {
      GetData();
      MakeFrame();
      SendFrame();                       //Send the data frame
      canSend = false;                   //Cannot send until ACK arrives
  }
  WaitForEvent();                        // Sleep until an event occurs
  if(Event(ArrivalNotification)          // An ACK has arrived
   {
      ReceiveFrame();                    //Receive the ACK frame
      canSend = true;
   }
}
```

- ***Receiver side***

```
while(true)                          //Repeat forever
{
  WaitForEvent();                    // Sleep until an event occurs
  if(Event(ArrivalNotification))     //Data frame arrives
  {
     ReceiveFrame();
     ExtractData();
     Deliver(data);                  //Deliver data to network layer
     SendFrame();                    //Send an ACK frame
  }
}
```

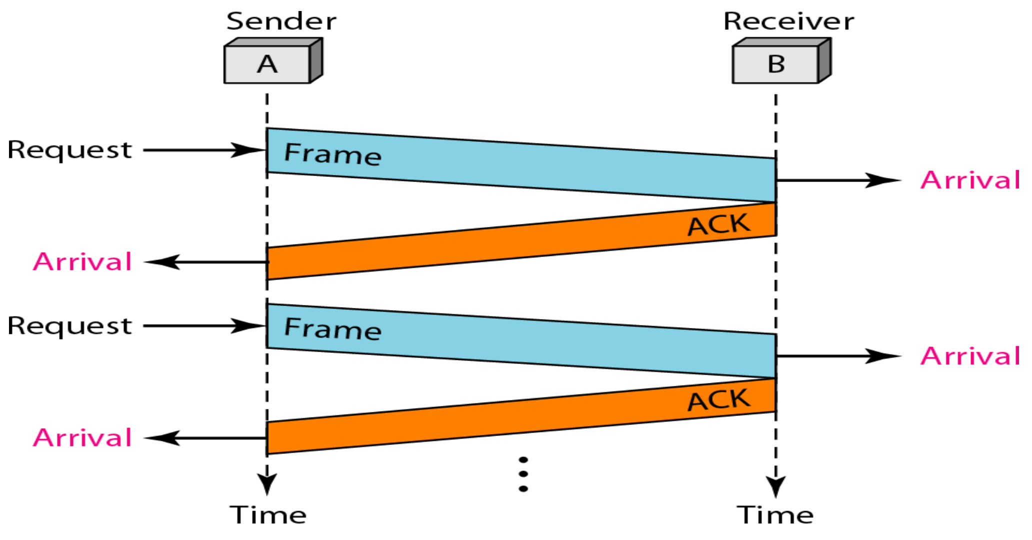


Fig:3. 20 Stop-and-Wait: Flow Diagram

Noisy channel protocols: stop and wait ARQ <u>Noisy</u>

Channel:

- Realistic

  Error can and will happen, Require error control

- Mechanisms:

  1. Stop-and-Wait ARQ
  2. Go-Back-N ARQ
  3. Selective Repeat ARQ

1. **<u>Stop-and-Wait</u> Automat<u>ic</u> Repe<u>a</u>t Request:**

The following transition may occur in Stop-and-Wait ARQ:

- The sender maintains a timeout counter. When a frame is sent, the sender starts the timeout counter.
- If acknowledgement of frame comes in time, the sender transmits the next frame in queue.
- If acknowledgement does not come in time, the sender assumes that either the frame or its acknowledgement is lost in transit. Sender retransmits the frame and starts the timeout counter.
- If a negative acknowledgement is received, the sender retransmits the frame.
- Sender keeps a copy of sent frame until successful delivery is ensured.
- Receiver responds with an ack when it successfully receives a frame.
- Both data and ack frames must be numbered.

Sequence Numbers

- The protocol specifies that frames need to be numbered. This is done by using sequence numbers. A field is added to the data frame to hold the sequence number of that frame. One important consideration is the range of the sequence numbers.

  For example, if we decide that the field is m bits long, the sequence numbers start from 0, go to $2m - 1$, and then are repeated.

Acknowledgment Number

- The acknowledgment numbers always announce the sequence number of the next frame expected by the receiver. For example, if frame 0 has arrived safe, the receiver sends an ACK frame with acknowledgment 1 (meaning frame 1 is expected next). If frame 1 has arrived safe, the receiver sends an ACK frame with acknowledgment 0 (meaning frame 0 is expected).
- In Stop-and-Wait ARQ the acknowledgment number always announces inmodulo-2 arithmetic the sequence number of the next frame expected.

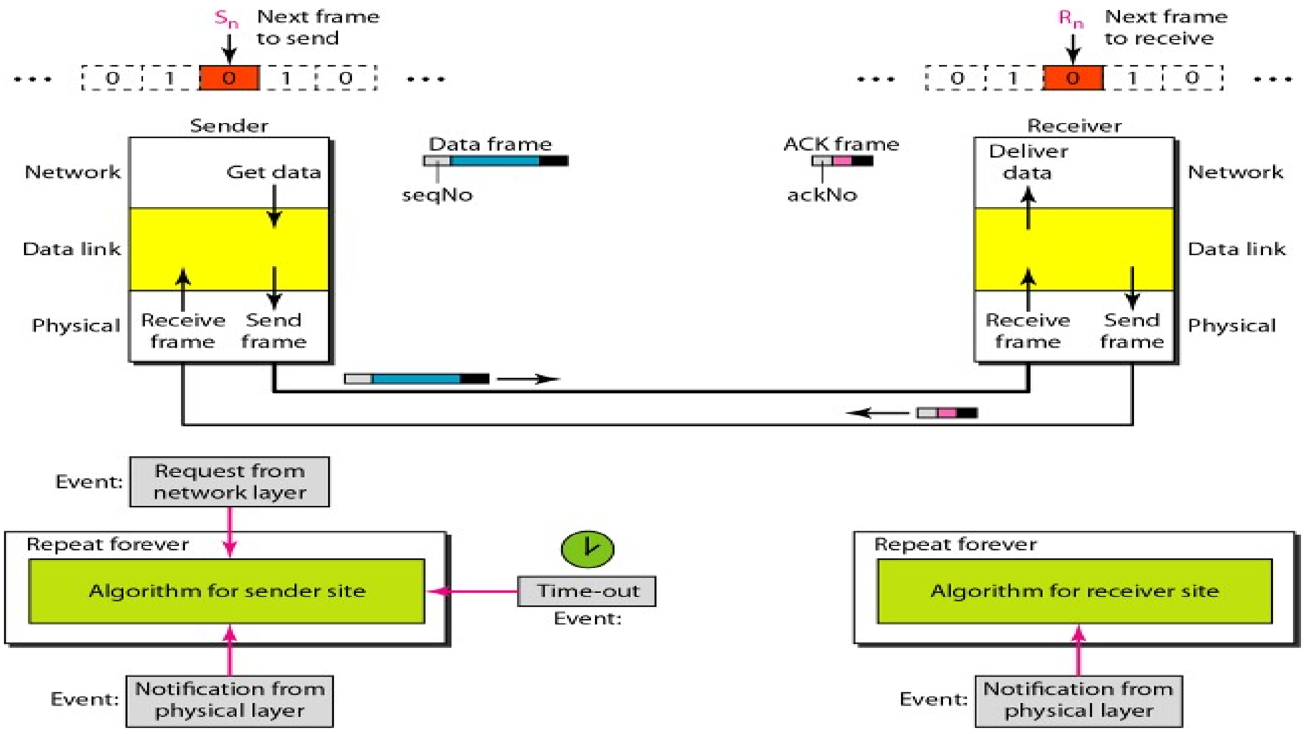
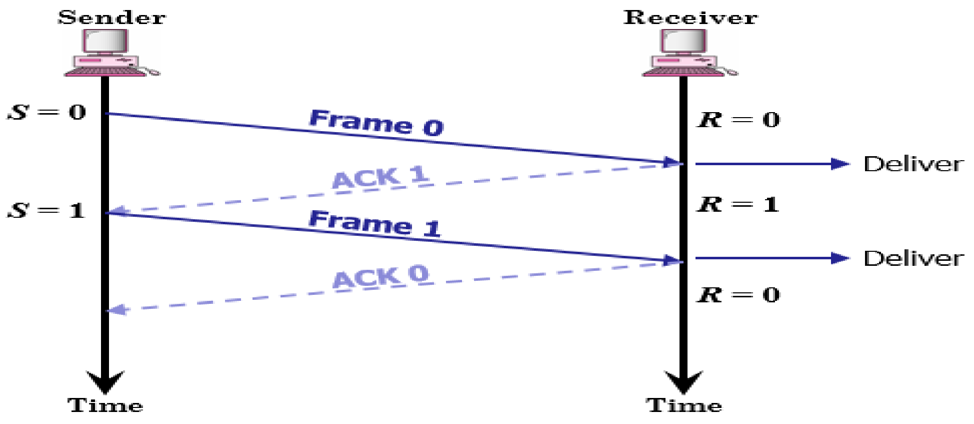


Fig: 3. 21 Stop-and-Wait ARQ

Flow Diagram: Normal Operation



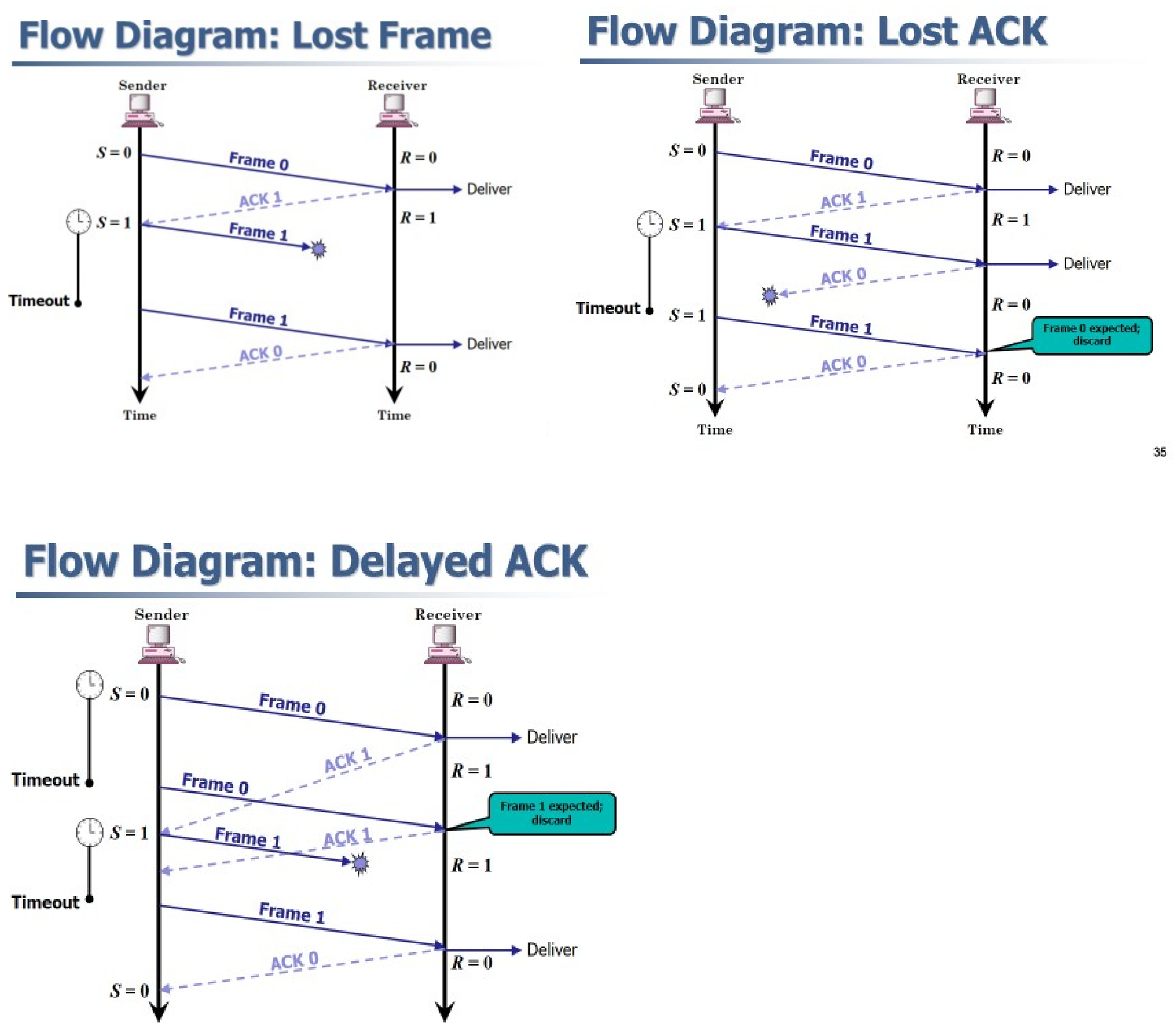


Fig: 3. 22 Frame Exchange

Pipelining

- *There is no pipelining in Stop-and-Wait ARQ because we need to wait for a frame to reach the destination and be acknowledged before the next frame can be sent. However, pipelining does apply to our next two protocols because several frames can be sent before we receive news about the previous frames.*
- *Pipelining improves the efficiency of the transmission if the number of bits in transition is large with respect to the bandwidth-delay product.*

Go-Back-N Automatic Repeat Request:

- In this protocol we can send several frames before receiving acknowledgments; we keep a copy of these frames until the acknowledgments arrive.
- These frames must be numbered differently. Frame numbers are called Sequence numbers
- Frames must be received in the correct order
- If a frame is lost, the lost frame and all of the following frames must be retransmitted

Sequence Numbers

- Frames from a sending station are numbered sequentially. However, because we need to include the sequence number of each frame in the header, we need to set a limit.
- If the header of the frame allows *m* bits for the sequence number, the sequence numbers range from 0 to $2^m - 1$.
- For example, if *m* is 4, the only sequence numbers are 0 through 15 inclusive. However, we can repeat the sequence. So the sequence numbers are

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

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

Sliding Window

The sliding window is an abstract concept that defines the range of sequence numbers that is the concern of the sender and receiver. The range which is the concern of the sender is called the send sliding window; the range that is the concern of the receiver is called the receive sliding window.

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

Sending Window

- Sending more than one frame at once requires sender to buffer multiple frames Known as "sending window". Any of these frames in the window can be lost.

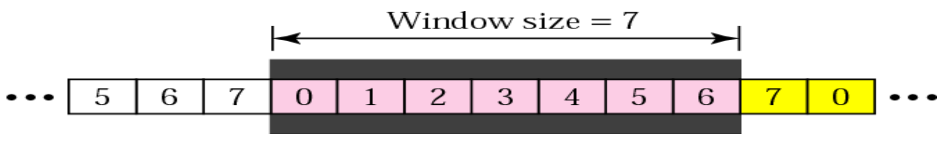


Fig: 3. 23 Sending Window

Sliding" Window"

- Once the first frames in the window is ACKed. ACKed frames are removed from the buffer. More frames are transmitted
- Result: The window slides to the right

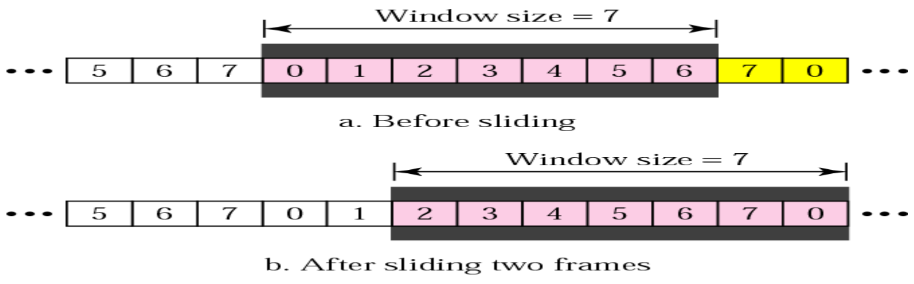


Fig: 3. 24 The window slides to the right

Receiving Window

Receiver expects one frame at a time.

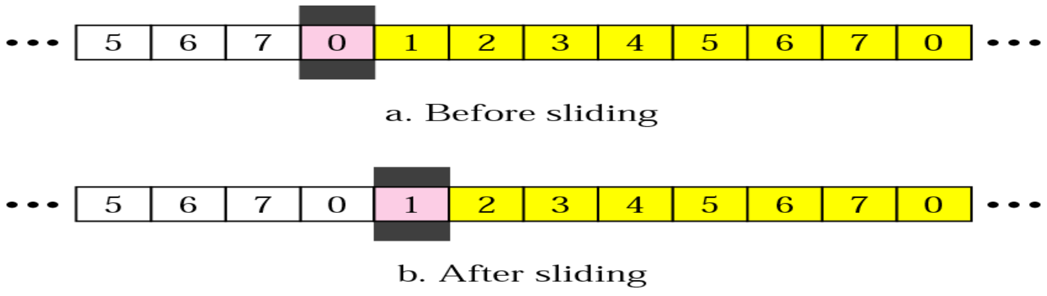
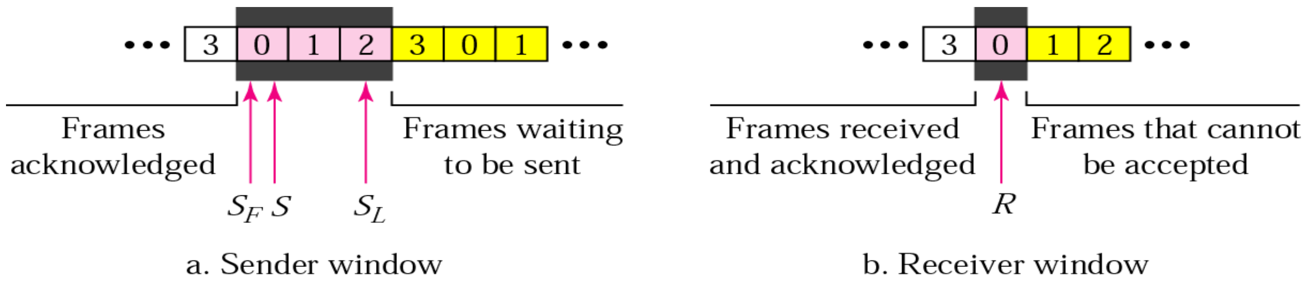


Fig: 3. 25 Receiving Window

- The **send window** is an abstract concept defining an **imaginary box of size $2^m - 1$** with three variables: S_f, S_n, and S_{size}.
- The **receive window** is an abstract concept defining an **imaginary box of size 1** with one single variable R_n. The window slides when a correct frame has arrived; sliding occurs one slot at a time.



3
0
1
2
3
0
1
Frames acknowledged
Frames waiting to be sent
S_F S
S_L
a. Sender window
3
0
1
2
Frames received and acknowledged
Frames that cannot be accepted
R
b. Receiver window

Fig: 3.26 Send vs. Receive Windows

Timers

- *Although there can be a timer for each frame that is sent, in our protocol we use only one. The reason is that the timer for the first outstanding frame always expires first; we send all outstanding frames when this timer expires.*

Acknowledgment

- *The receiver sends a positive acknowledgment if a frame has arrived safe and sound and in order. If a frame is damaged or is received out of order, the receiver is silent and will discard all subsequent frames until it receives the one it is expecting.*

Resending a Frame

- *When the timer expires, the sender resends all outstanding frames*

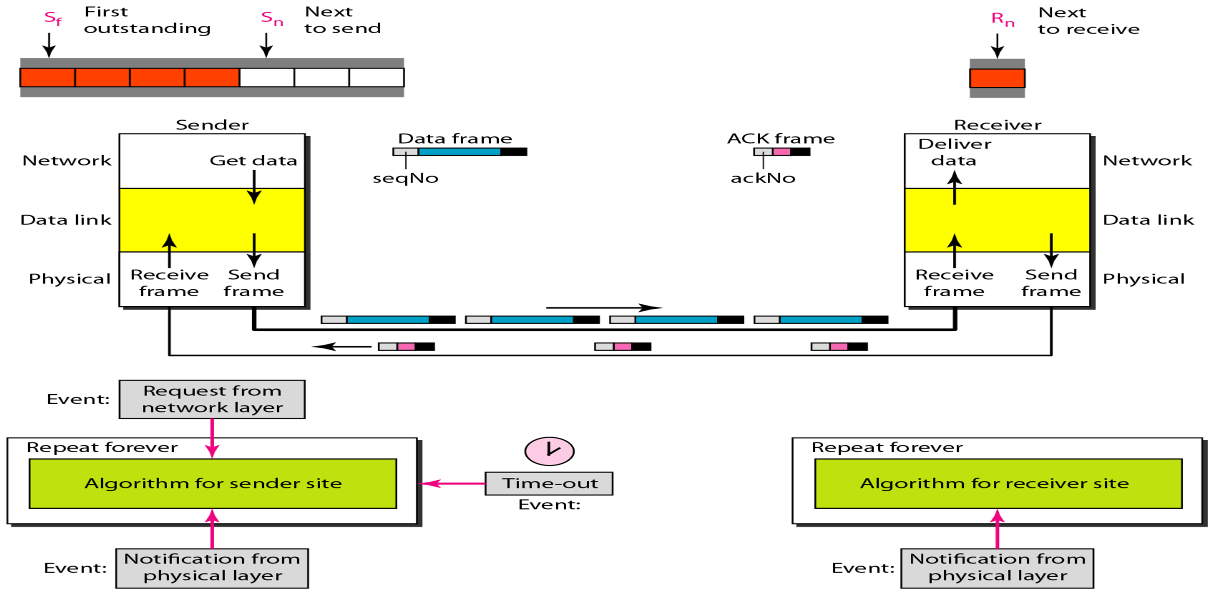


Fig: 3.27 Design of Go-Back-N ARQ

Go-Back-N: Window Sizes

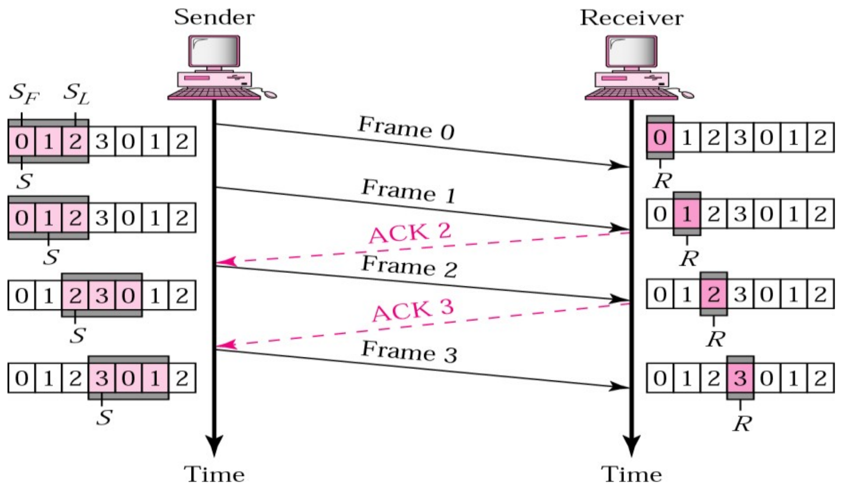
- For m-bit sequence numbers. Send window size: at most 2^m

→ Up to 2^m-1 frames can be sent without ACK

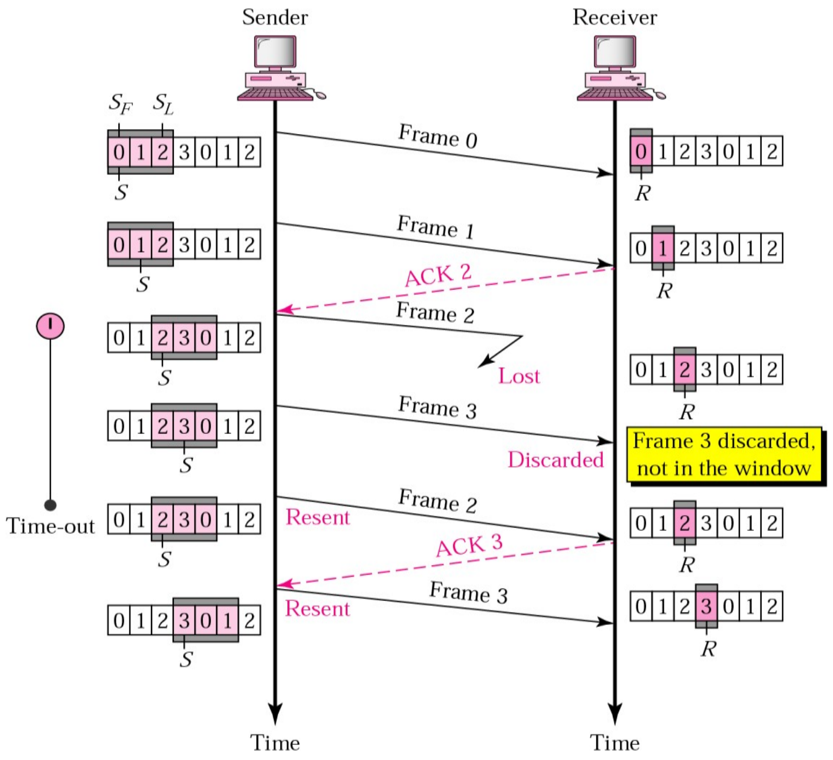
- Receive window size: 1

→ Frames must be received in order

Go-Back-N: Normal Operation



Go-Back-N: Lost Frame



Window size for Go-Back-N ARQ

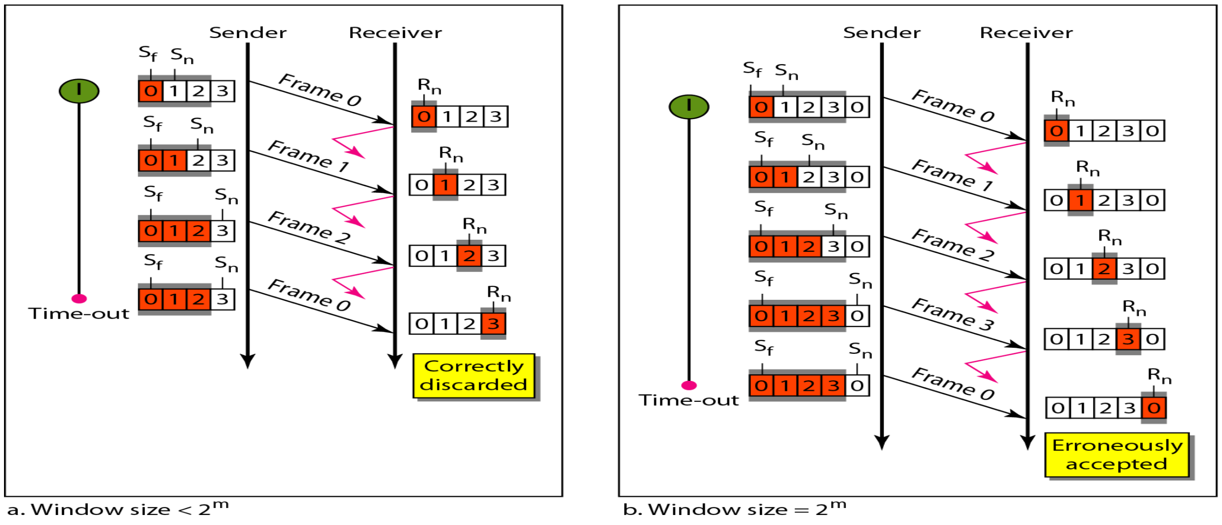


Fig: 3. 28 Frame Exchange

- In Go-Back-N ARQ, the size of the send window must be less than 2^m; the size of the receiver window is always 1.

Selective Repeat Automatic Repeat Request:

- For noisy links, there is another mechanism that does not resend *N* frames when just one frame is damaged; only the damaged frame is resent. This mechanism is called Selective Repeat ARQ. It is more efficient for noisy links, but the processing at the receiver is more complex.

Windows

- The Selective Repeat Protocol also uses two windows: a send window and a receive window.
- Sender and receiver share window space equally

  For *m*-bit sequence numbers
- Send window: up to 2^{m-1}
- Receive window: up to 2^{m-1}

  For example, if $m = 4$, the sequence numbers go from 0 to 15, but the size of the window is just 8 (it is 15 in the *Go-Back-N* Protocol). The smaller window size means less *efficiency in filling the pipe, but the fact that there are fewer duplicate frames can compensate for this.*

Send Window

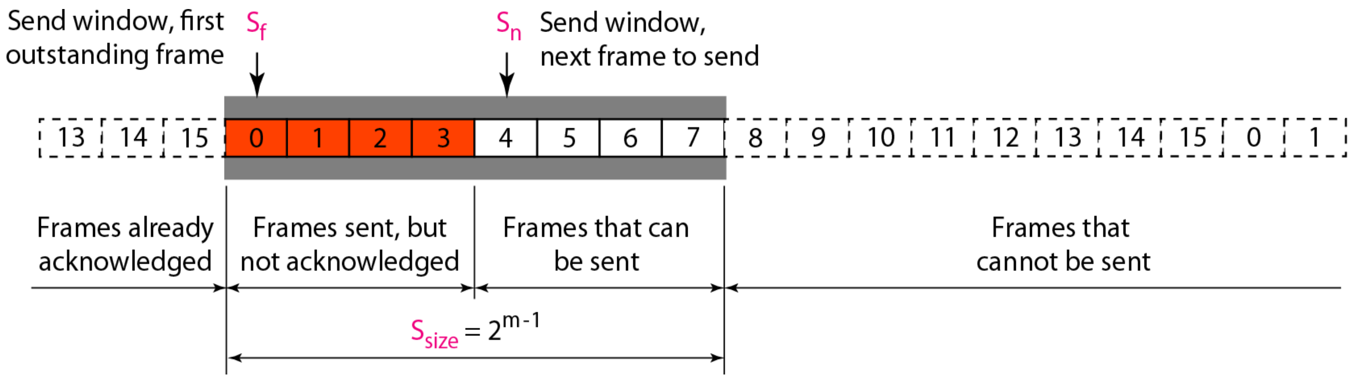

Send window, first outstanding frame
S_f
S_n
Send window, next frame to send
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
Frames already acknowledged
Frames sent, but not acknowledged
Frames that can be sent
Frames that cannot be sent
$S_{size} = 2^{m-1}$

Fig: 3.29 Send Window

Receive Window

- *The Selective Repeat Protocol allows as many frames as the size of the receive window to arrive out of order and be kept until there is a set of in-order frames to be delivered to the network layer. All the frames in the send frame can arrive out of order and be stored until they can be delivered.*

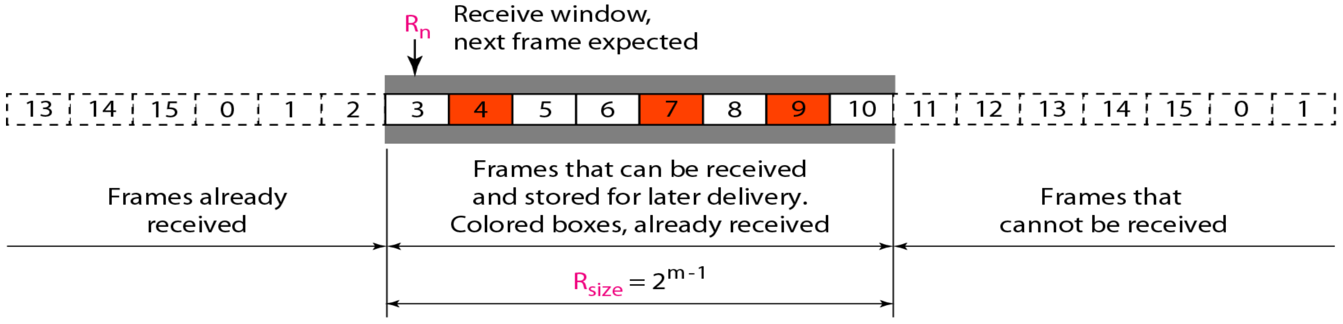

R_n
Receive window, next frame expected
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
Frames already received
Frames that can be received and stored for later delivery. Colored boxes, already received
Frames that cannot be received
$R_{size} = 2^{m-1}$

Fig: 3.30 Receive Window

Window Sizes

- *The size of the sender and receiver windows must be at most one half of 2m.*

For an example, we choose m = 2, which means the size of the window is 2m/2, or 2. Compares a window size of 2 with a window size of 3. If the size of the window is 2 and all acknowledgments are lost, the timer for frame 0 expires and frame 0 is resent. However, the window of the receiver is now expecting frame 2, not frame 0, so this duplicate frame is correctly discarded. When the size of the window is 3 and all acknowledgments are lost, the sender sends a duplicate of frame 1. However, this time, the window of the receiver expects to receive frame 0 (0 is part of the window), so it accepts frame 0, not as a duplicate, but as the first frame in the next cycle. This is clearly an error.

Design of Selective Repeat ARQ

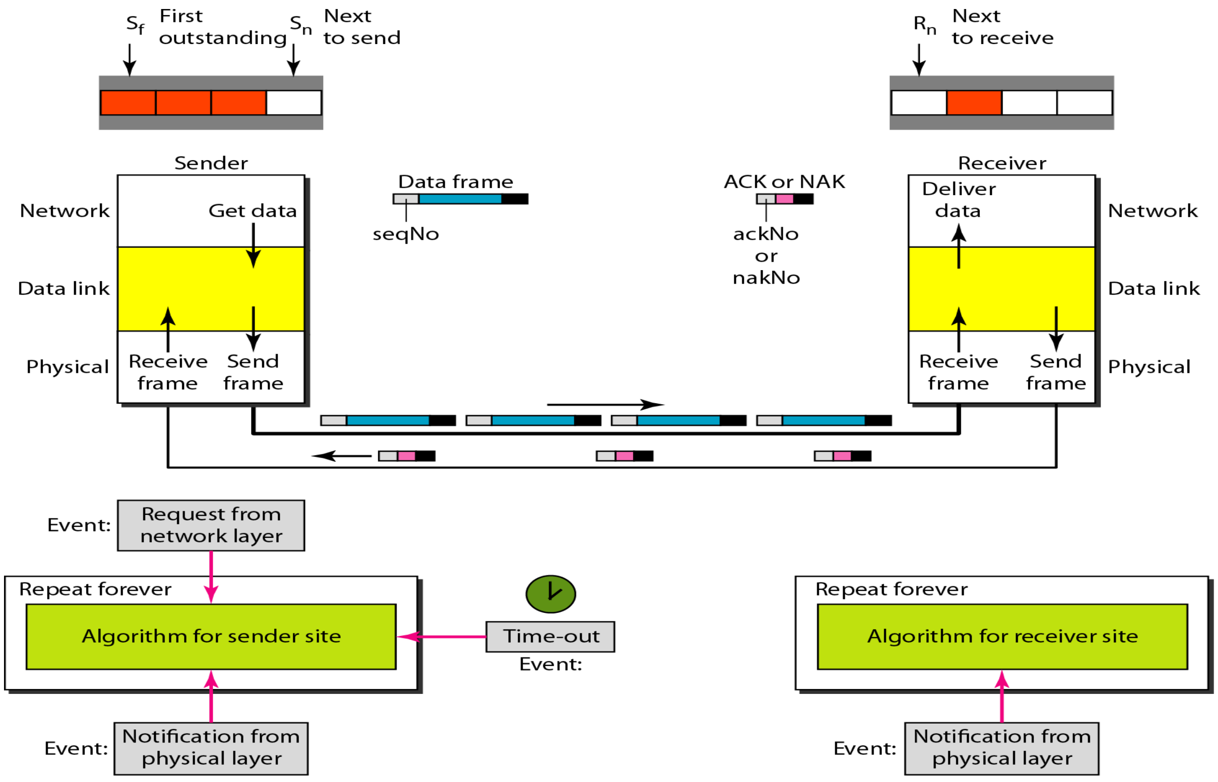


Fig: 3. 31 Design of Selective Repeat ARQ

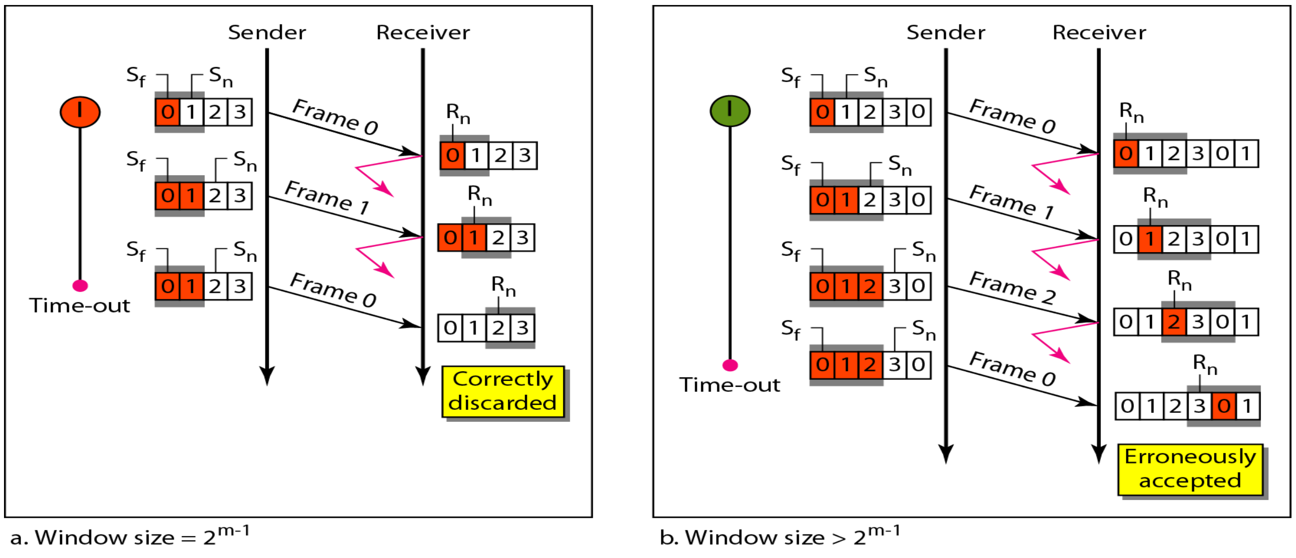


Fig: 3. 32 Frame Exchange

Negative ACK

Used by receiver to indicate missing frame

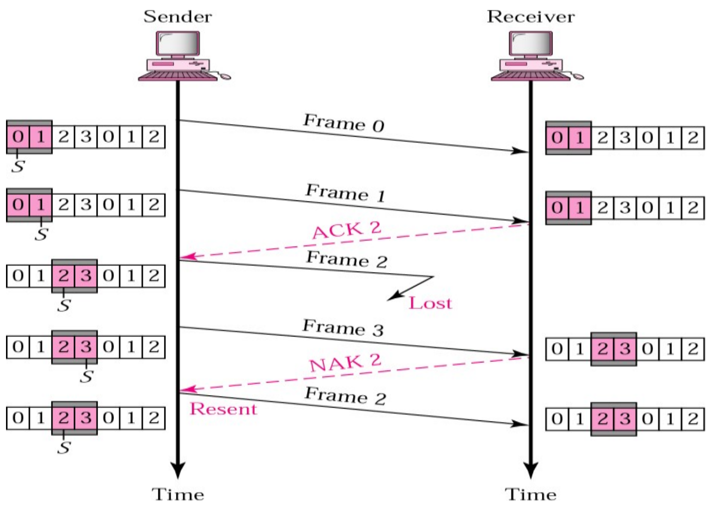


Fig:3. 33 Negative ACK

How Selective Repeat behaves

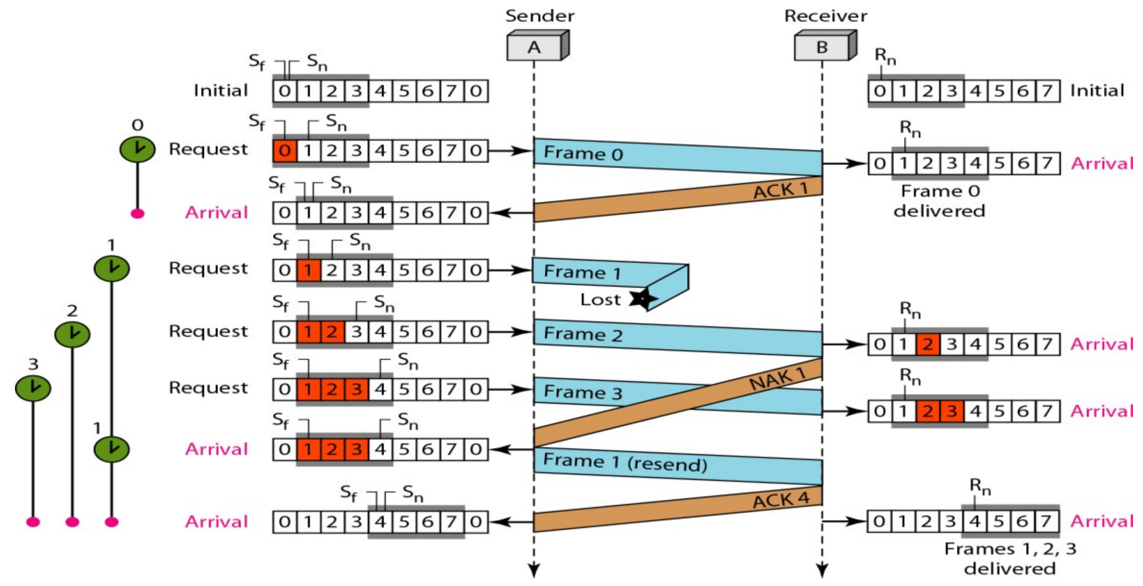


Fig: 3. 34 Frame Exchange

Piggybacking

The three protocols we discussed in this section are all unidirectional: data frames flow in only one direction although control information such as ACK and NAK frames can travel in the other direction. In real life, data frames are normally flowing in both directions: from node A to node B and from node B to node A. This means that the control information also needs to flow in both directions. A technique called **piggybacking** issued to improve the efficiency of the bidirectional protocols.

When a frame is carrying data from A to B, it can also carry control information about arrived (or lost) frames from B; when a frame is carrying data from B to A, it can also carry control information about the arrived (or lost) frames from A.

Design of piggybacking in Go-Back-N ARQ

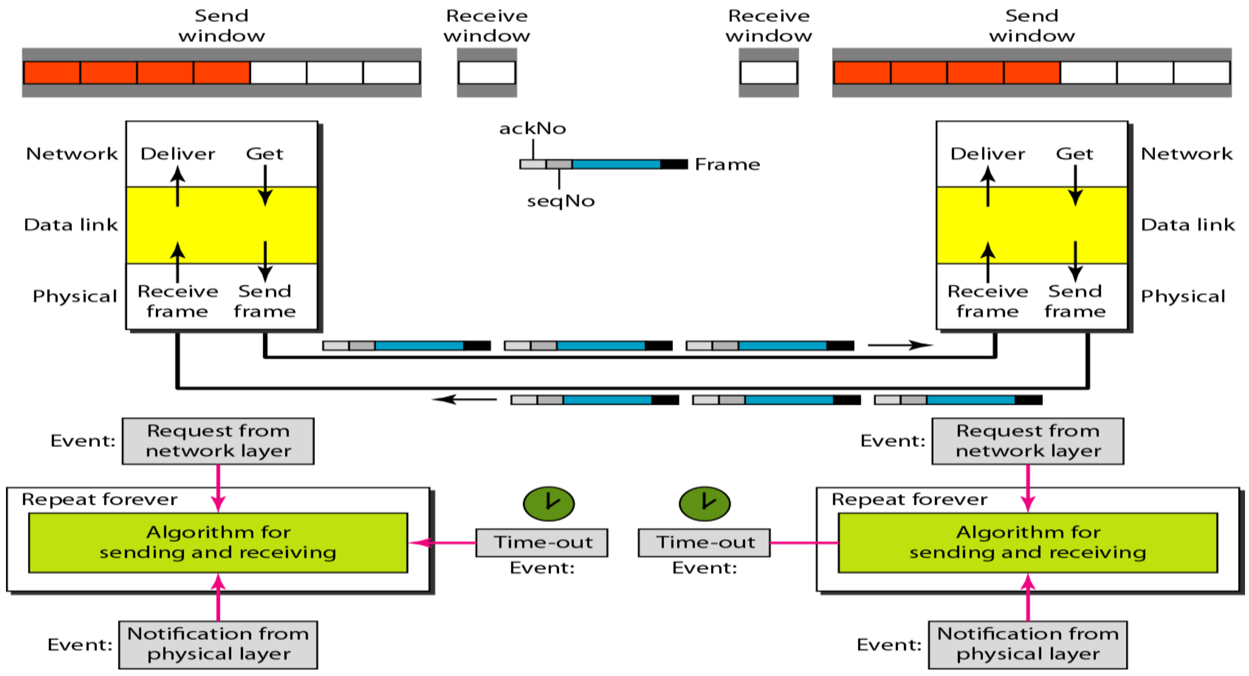


Fig: 3. 35 Design of piggybacking in Go-Back-N ARQ

- An important point about piggybacking is that both sites must use the same algorithm. This algorithm is complicated because it needs to combine two arrival events into one.

HDLC protocol: Frame formats

High -Level Data Link Control (HDLC):

- High-level Data Link Control (HDLC) is a bit-oriented protocol for communication over point-to-point and multipoint links.
- HDLC is a synchronous Data Link layer bit-oriented protocol developed by the International Organization for Standardization (ISO).
- HDLC provides both connection-oriented and connectionless service.
- HDLC defines a Layer 2 framing structure that allows for flow control and error control through the use of acknowledgments.
- It implements the ARQ mechanisms.

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*

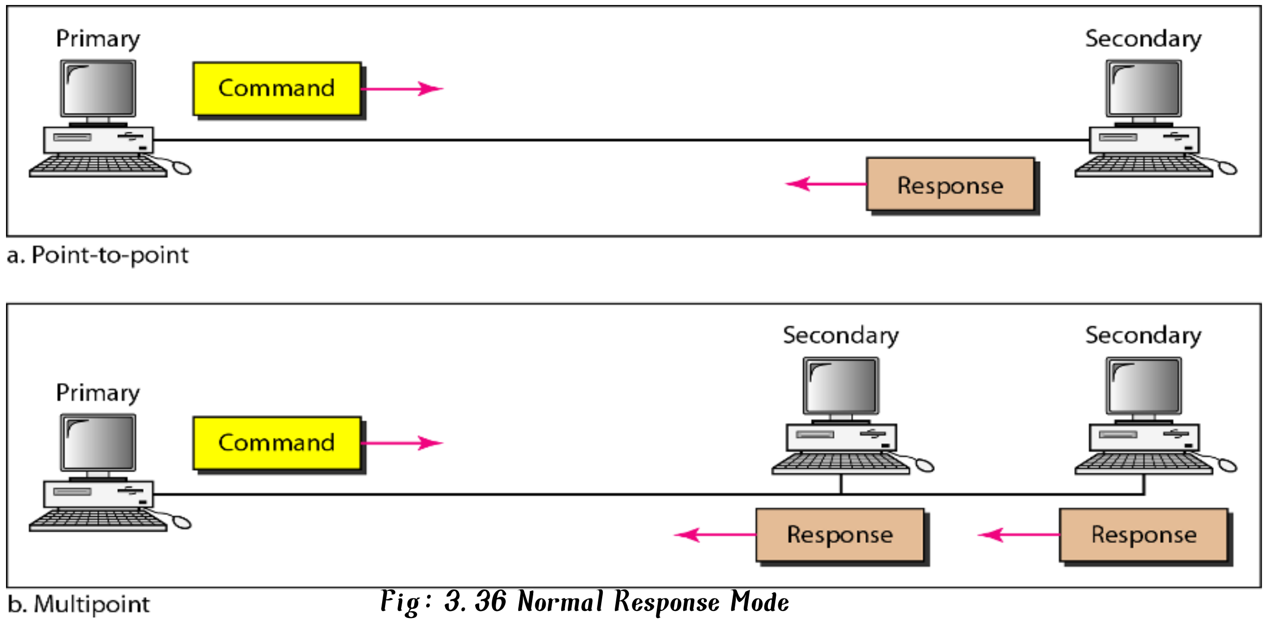


Fig: 3. 36 Normal Response Mode

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).*

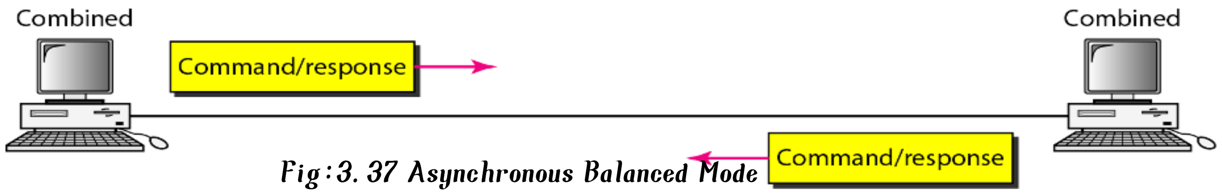


Fig:3. 37 Asynchronous Balanced Mode

HDLC defines three types of frames:

2. Information frames :(I-frames)
3. Supervisory frames (S-frames)
4. 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.

Frame Format

- Each frame in HDLC may contain up to six fields, a beginning flag field, an address field, a control field, an information field, a frame check sequence (FCS) field, and an ending flag field. In multiple-frame transmissions, the ending flag of one frame can serve as the beginning flag of the next frame

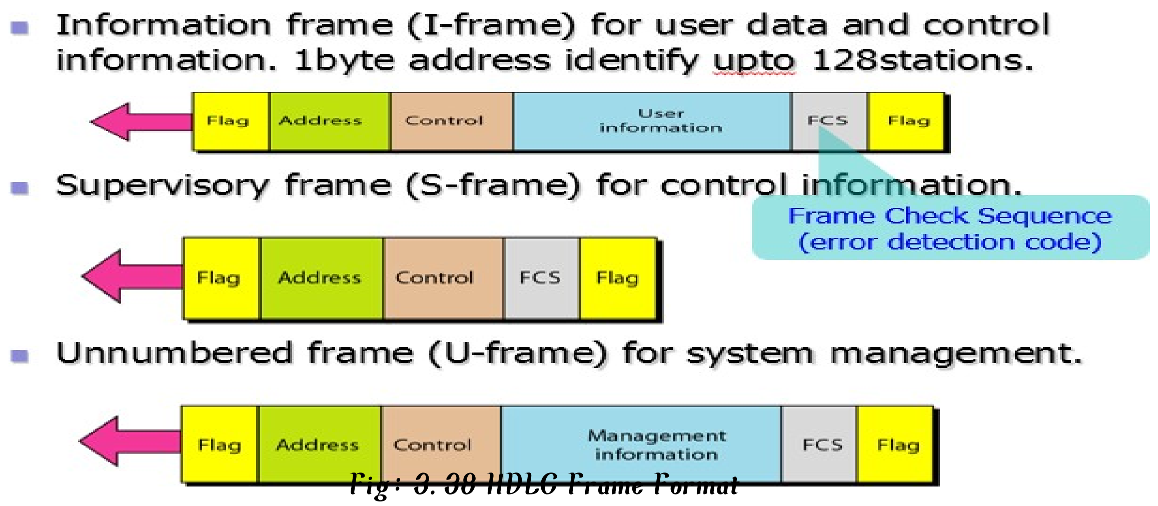

Information frame (I-frame) for user data and control information. 1byte address identify upto 128stations.
Flag
Address
Control
User information
FCS
Flag
Supervisory frame (S-frame) for control information.
Frame Check Sequence (error detection code)
Flag
Address
Control
FCS
Flag
Unnumbered frame (U-frame) for system management.
Flag
Address
Control
Management information
FCS
Flag

Fig: 3.30 HDLC Frame Format

Fields

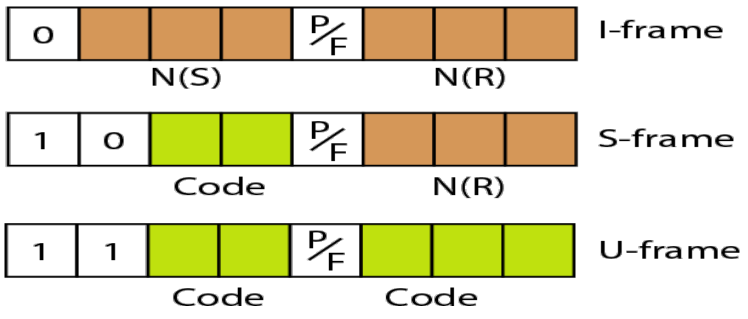
- **Flag field.** The flag field of an HDLC frame is an 8-bit sequence with the bit pattern01111110 that identifies both the beginning and the end of a frame and serves as a synchronization pattern for the receiver.

- ***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.*
- ***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.*

Control Field

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



N(S) - Frame sequence number,

N(R) - Ack sequence number P/F

Poll (primary → secondary)

Final (secondary → primary)

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 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 and 7; but in the extension format, in which the control field is 2 bytes, this field is larger.
- 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**. 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.

S Frame:

- 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. 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. It acknowledges the receipt of a frame or group of frames, and it announces that the receiver is busy and cannot receive more frames.

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.

Selective reject (SREJ).

If the value of the code subfield is 11, it is an SREJ S-frame. This is a NAK frame used in Selective Repeat ARQ. Note that the HDLC Protocol uses the term selective reject instead of selective repeat. The value of N(R) is the negative acknowledgment number.

- The fifth field in the control field is the P/F bit as discussed before.
- The next 3 bits, called N(R), correspond to the ACK or NAK value.

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.
- 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

Implementing piggybacking in HDLC

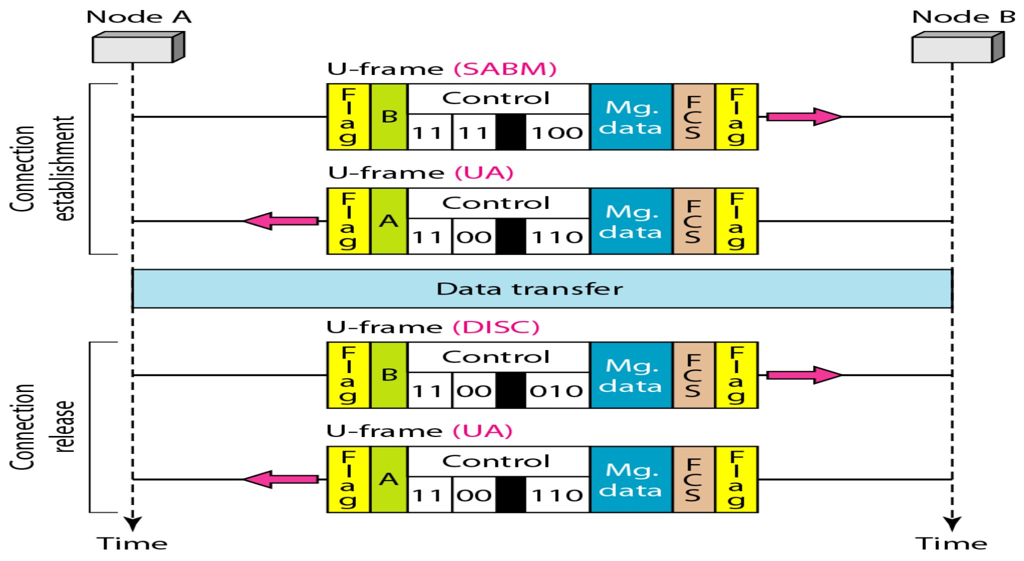


Fig: 3. 39 Connection/Disconnection

- *It shows how V-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 (VA) frame. After these two exchanges, data can be transferred between the two nodes (not shown in the figure). 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).*

Data Transfer: Without Error Data Transfer: With Error

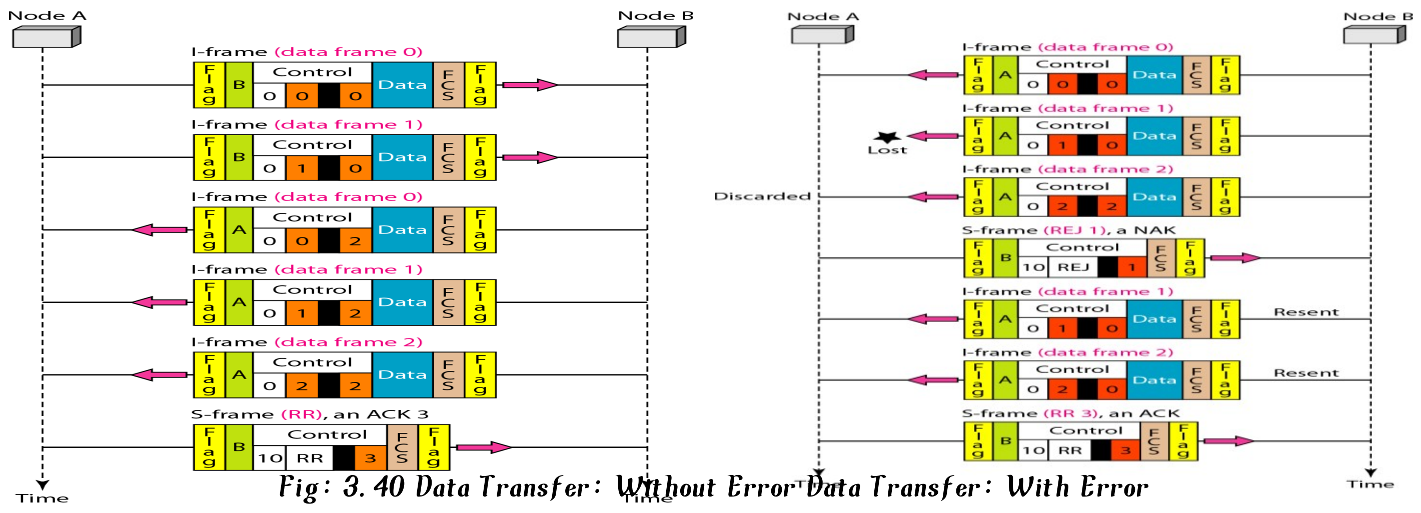


Fig: 3. 40 Data Transfer: Without Error Data Transfer: With Error

Point to point protocol (PPP): framing, transition phase

POINT-TO-POINT PROTOCOL:

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.

PPP is a byte-oriented protocol using byte stuffing with the escape byte 01111101. As a byte- oriented Protocol, the flag in PPP is a byte and needs to be escaped whenever it appears in the data section of the frame. The escape byte is 01111101, which means that every time the flag like pattern appears in the data, this extra byte is stuffed to tell the receiver that the next byte is not a flag.

PPP provides several services:

1. PPP defines the format of the frame to be exchanged between devices.
2. PPP defines how two devices can negotiate the establishment of the link and the exchange of data.
3. PPP defines how network layer data are encapsulated in the data link frame.
4. PPP defines how two devices can authenticate each other.
5. PPP provides multiple network layer services supporting a variety of network layer protocols.
6. PPP provides connections over multiple links.
7. PPP provides network address configuration.

On the other hand, to keep PPP simple, several services are missing:

1. PPP does not provide flow control.
2. 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;
3. PPP does not provide a sophisticated addressing mechanism to handle frames in a multipoint configuration.

Framing:

PPP is a byte-oriented protocol.

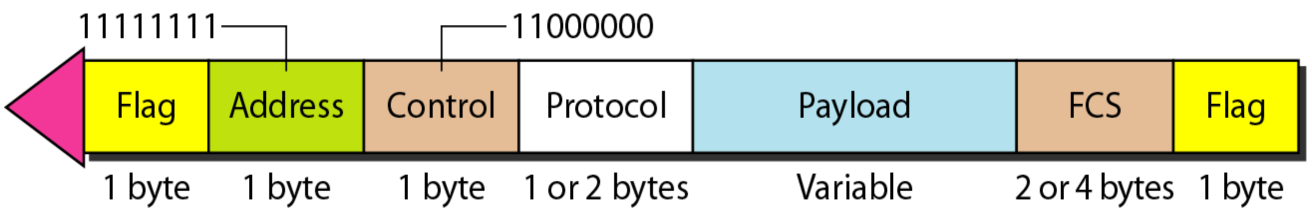


Fig: 3.41 PPP Frame format

- ***Flag field***: *Flag field marks the beginning and end of the PPP frame. Flag byte is 01111110. (1 byte).*
- ***Address field***: *This field is of 1 byte and is always 11111111. This address is the broadcast address i.e. all the stations accept this frame.*
- ***Control field***: *This field is also of 1 byte. This field uses the format of the U-frame (unnumbered) in HDLC. The value is always 00000011 to show that the frame does not contain any sequence numbers and there is no flow control or error control.*
- ***Protocol field***: *This field specifies the kind of packet in the data field i.e. what is being carried in payload(data) field.*
- ***Payload field***: *Its length is variable. If the length is not negotiated using LCP during line set up, a default length of 1500 bytes is used. It carries user data or other information.*
- ***FCS field***: *The frame checks sequence. It is either of 2 bytes or 4 bytes. It contains the checksum.*

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

Transition Phases in PPP

The PPP connection goes through different states.

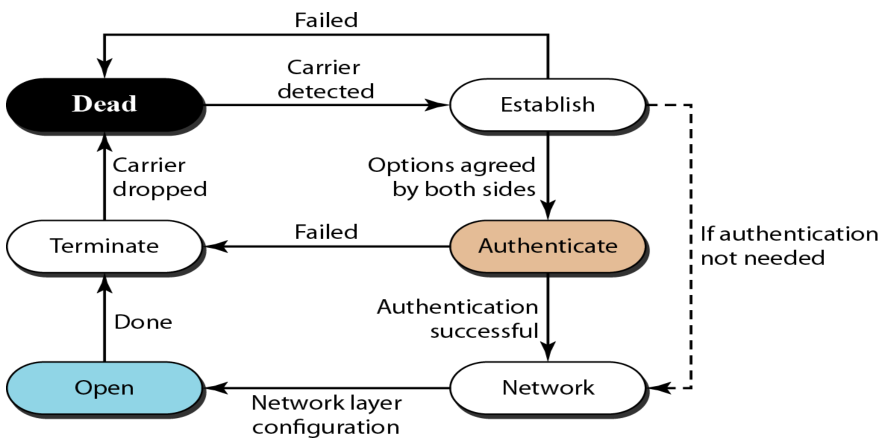

Failed
Carrier detected
Dead
Establish
Carrier dropped
Options agreed by both sides
Terminate
Failed
Authenticate
If authentication not needed
Done
Authentication successful
Open
Network layer configuration
Network

Fig: 3.42 Transition Phases in PPP

- **Dead**: *In dead phase the link is not used. There is no active carrier and the line is quiet.*
- **Establish**: *Connection goes into this phase when one of the nodes start communication. In this phase, two parties negotiate the options. If negotiation is successful, the system goes into authentication phase or directly to networking phase. LCP packets are used for this purpose.*
- **Authenticate**: *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 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.*
- **Open**: *In this phase, data transfer takes place. The connection remains in this phase until one of the endpoints wants to end the connection.*
- **Terminate**: *In this phase connection is terminated.*