Branch **: CSE**

Subject **: Computer Networks**

UNIT – II

Data link layer – design issues – Services - Framing - Error Control - Flow Control - Error detection and correction codes - data link layer protocols -Simplex Protocol – Sliding window Protocols - Medium Access control sub layer – Channel allocation problem – Multiple Access protocols – ALOHA – CSMA Protocols - Collision-Free Protocols - Limited-Contention Protocols - Wireless LANs - 802.11 Architecture - 802.16 Architecture – Data link layer Switching - Uses of Bridges - Learning Bridges - Spanning Tree Bridges - Repeaters, Hubs, Bridges, Switches, Routers, and Gateways - Virtual LANs.

What is DLL (Data Link Layer)?

The Data Link Layer is the second layer in the OSI model, above the Physical Layer, which ensures that the error free data is transferred between the adjacent nodes in the network. It breaks the datagram passed down by above layers and converts them into frames ready for transfer. This is called **Framing.**

It provides two main functionalities

- Reliable data transfer service between two peer network layers
- Flow Control mechanism which regulates the flow of frames such that data congestion is not there at slow receivers due to fast senders.

THE DATA LINK LAYER DESIGN ISSUES

FUNCTIONS

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

The two main functions of the data link layer are:

1. **Data Link Control (DLC)**: It deals with the design and procedures for communication b/w nodes: node-to-node communication.

2. **Media Access Control (MAC)**: It explains how to share the link.

1. DATA LINK CONTROL (DLC):

Data link control functions includes

(1) Framing.

(2) Error Control.

(3) Flow Control.

(1) FRAMING

The frame contains

1. Frame header
2. Payload field for holding packet
3. Frame trailer

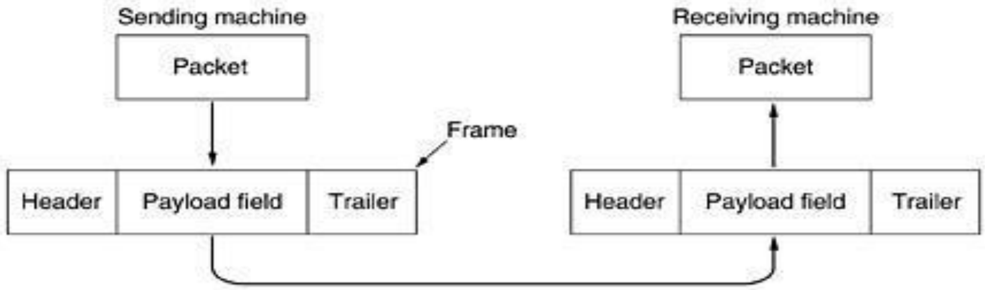


Figure 1.1 Relationships between Packets and Frames

Services provided to the network layer

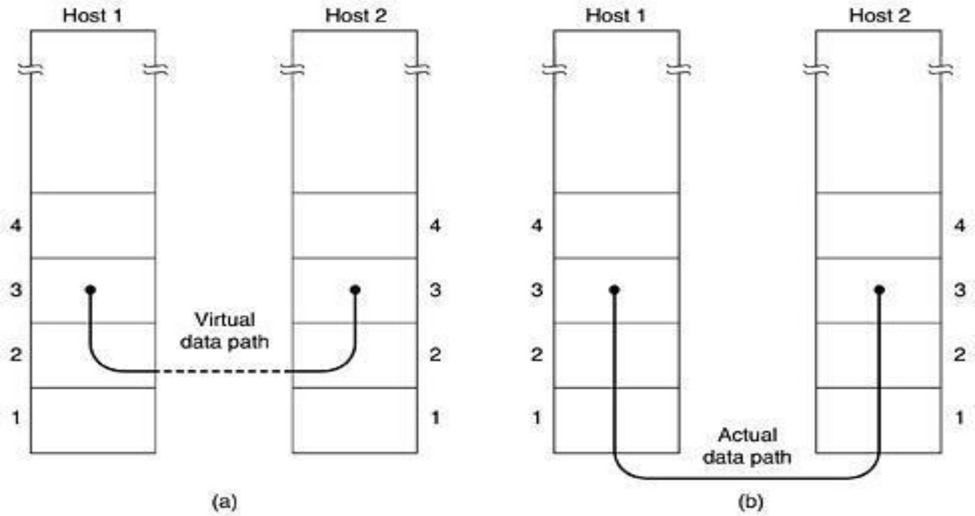


Figure 1.2 (a) Virtual communication. (b) Actual communication.

Transferring data from the network layer on the source machine to the network layer on the destination machine. 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**

   - Source machine sends independent frames to destination machine having destination machine acknowledge them
   - No logical connection
   - Used when error rate is very low
   - Good for real-time traffic (voice)

2. **Acknowledged connectionless service**

   - No logical connection
   - Each frame sent is individually acknowledged
   - Useful over unreliable channels (i.e. wireless systems)

3. **Acknowledged connection-oriented service**

   - Source and destination machines establish a connection before any data are transferred
   - Each frame is numbered
   - DLL guarantees that...
     - Each frame is received
     - Each frame is received exactly once
     - Each frame is received in the right order

3 PHASES

When connection-oriented service is used, transfers go through three distinct phases

1. Connection established
2. Frames are transmitted
3. Connection released

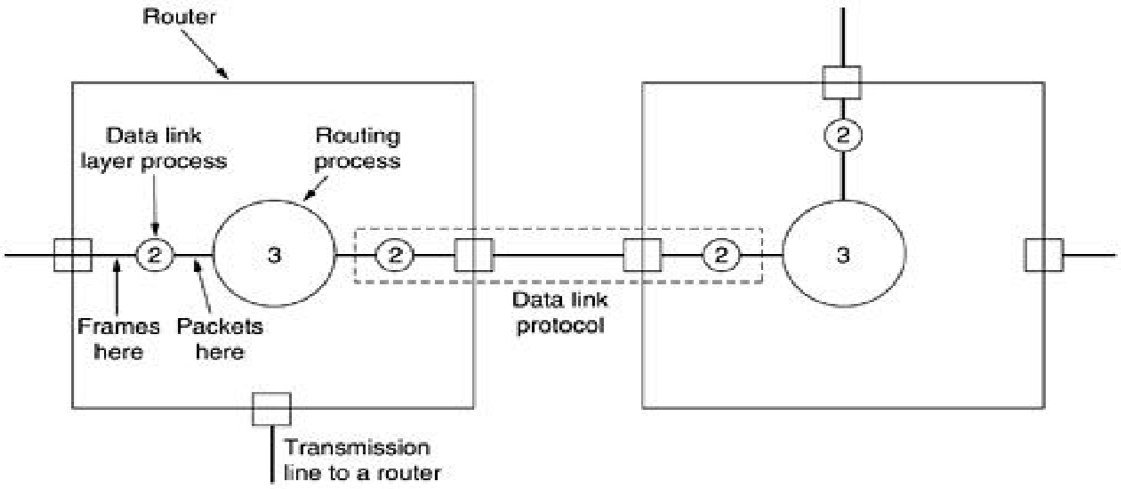


Figure 1.3 Placement of the data link Protocol

- Consider a typical example: a WAN subnet consisting of routers connected by point-to-point leased telephone lines.
- When a frame arrives at a router, the hardware checks it for errors, and then passes the frame to the data link layer software.
- The data link layer software checks to see if this is the frame expected, and if so, gives the packet contained in the payload field to the routing software.
- The routing software then chooses the appropriate outgoing line and passes the packet back down to the data link layer software, which then transmits it. The flow over two routers is shown in **Fig. 1-3.**

FRAMING

Breaking the bit stream up into frames is more difficult than it at first appears. One way to achieve this framing is to insert time gaps between frames, much like the spaces between words in ordinary text. However, networks rarely make any guarantees about timing, so it is possible these gaps might be squeezed out or other gaps might be inserted during transmission.

There are four methods:

1. Character count.

2. Flag bytes with byte stuffing.

3. Starting and ending flags, with bit stuffing.

4. 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 Fig. 3-4(a) for four frames of sizes 5, 5, 8, and 8 characters, respectively.

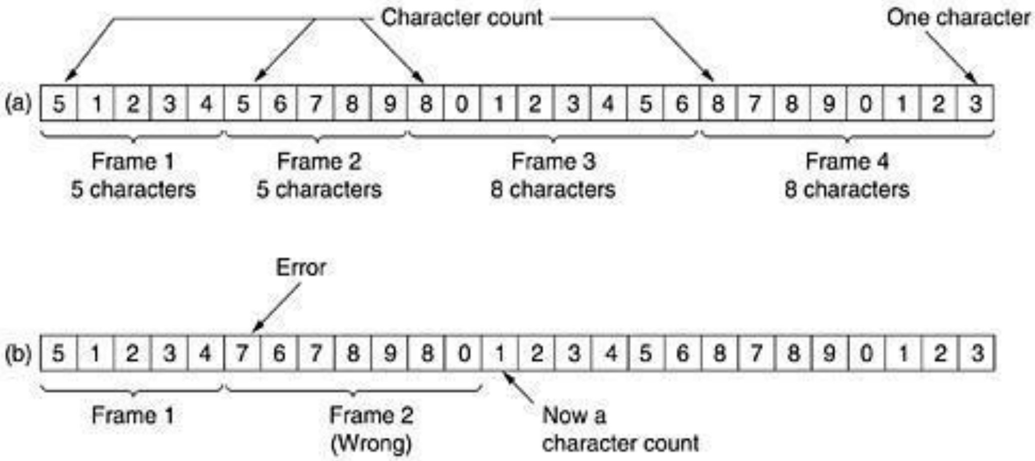


Figure 3-4. A character stream. (a) Without errors. (b) With one error.

Explanation (Figure 3-4.(a) A character stream Without errors.)

- 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 Fig. 3-4(a) for four frames of sizes 5, 5, 8, and 8 characters, respectively.
- The trouble with this algorithm is that the count can be garbled by a transmission error.

Explanation (Figure 3-4.(b) A character stream with errors.)

- For example, if the character count of 5 in the second frame of Fig. 3-4(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 approach

- In a character-oriented approach, data to be carried are 8-bit characters.
- The header, which normally carries the source and destination addresses and other control information.
- Trailer 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.

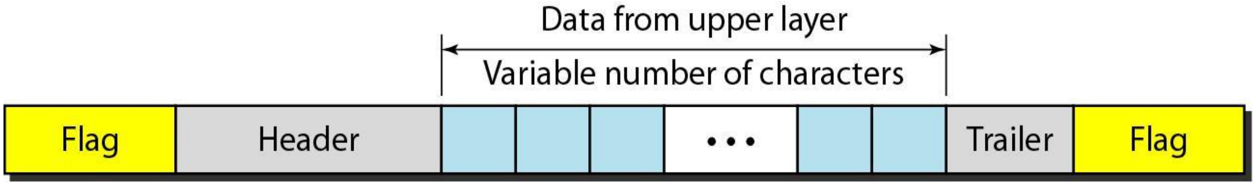


Figure: shows the format of a frame in a character-oriented protocol

Advantage:

1. Simple framing method.
2. Character-oriented framing was popular when only text was exchanged by the data Link layers.
3. The flag could be selected to be any character not used for text communication.

Disadvantage:

1. Even if with checksum, the receiver knows that the frame is bad there is no way to tell where the next frame starts.
2. Asking for retransmission doesn’t help either because the start of the retransmitted frame is not known.
3. Hence No longer used.

Starting and ending character with byte stuffing

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

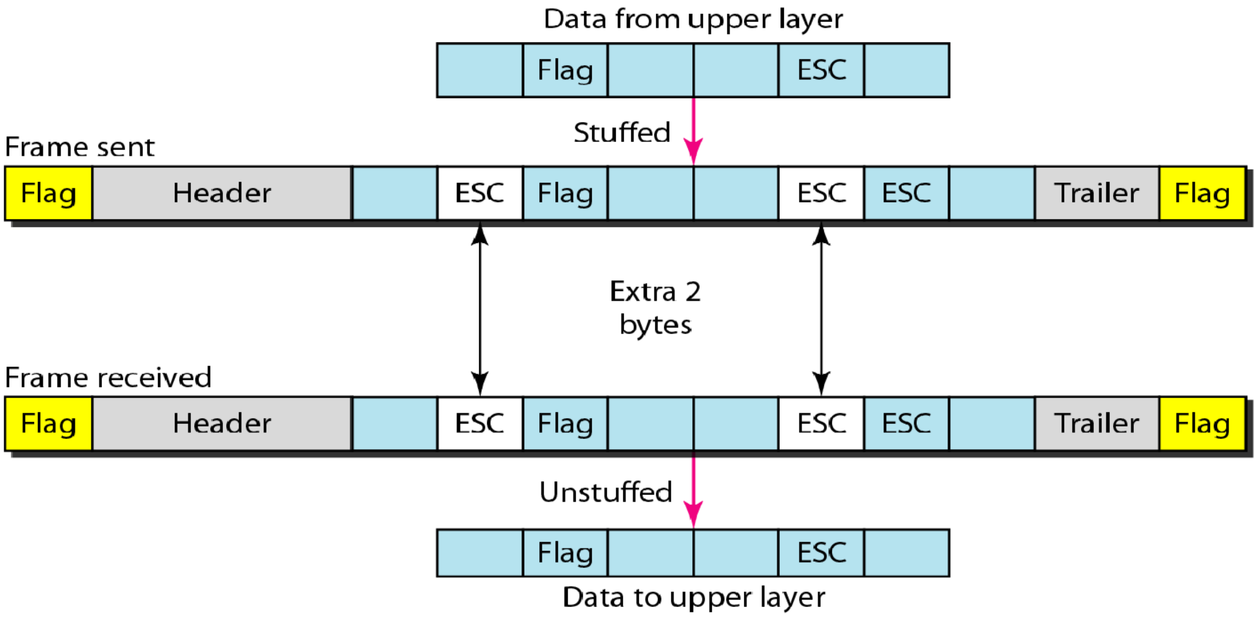


Figure : Byte stuffing and unstuffing

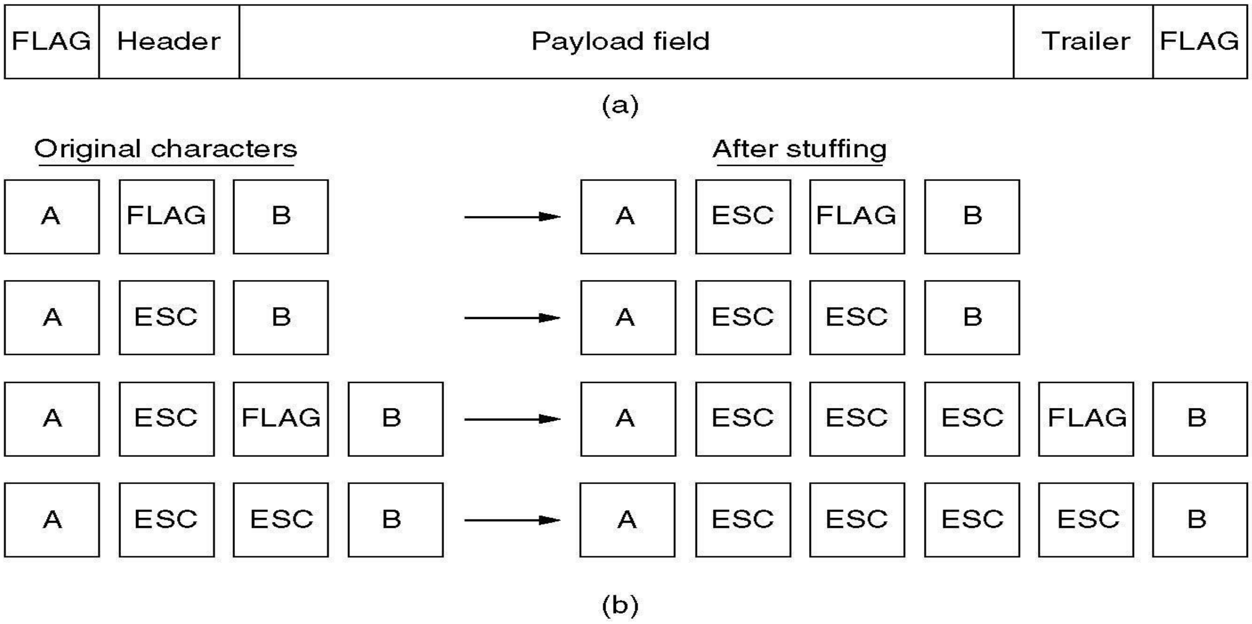


Fig: Framing with byte stuffing

Problem: fixed character size: assumes character size to be 8 bits: can’t handle heterogeneous environment.

Bit-Oriented framing approach

- Bit stuffing is the process of adding one extra 0 whenever five consecutive 1’s follow a 0 in the data, so that the receiver does not mistake the pattern 0111110 for a flag.
- Most protocols use a special 8-bit pattern flag 01111110 as the delimiter to define the beginning and the end of the frame, as shown in Figure below
- This flag can create the same type of problem. 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.

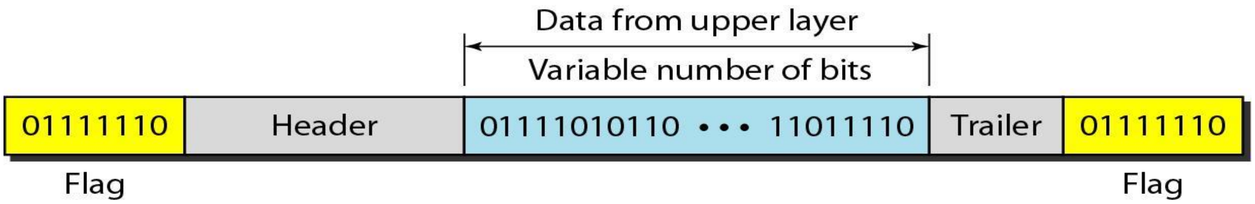


Figure (a)

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.

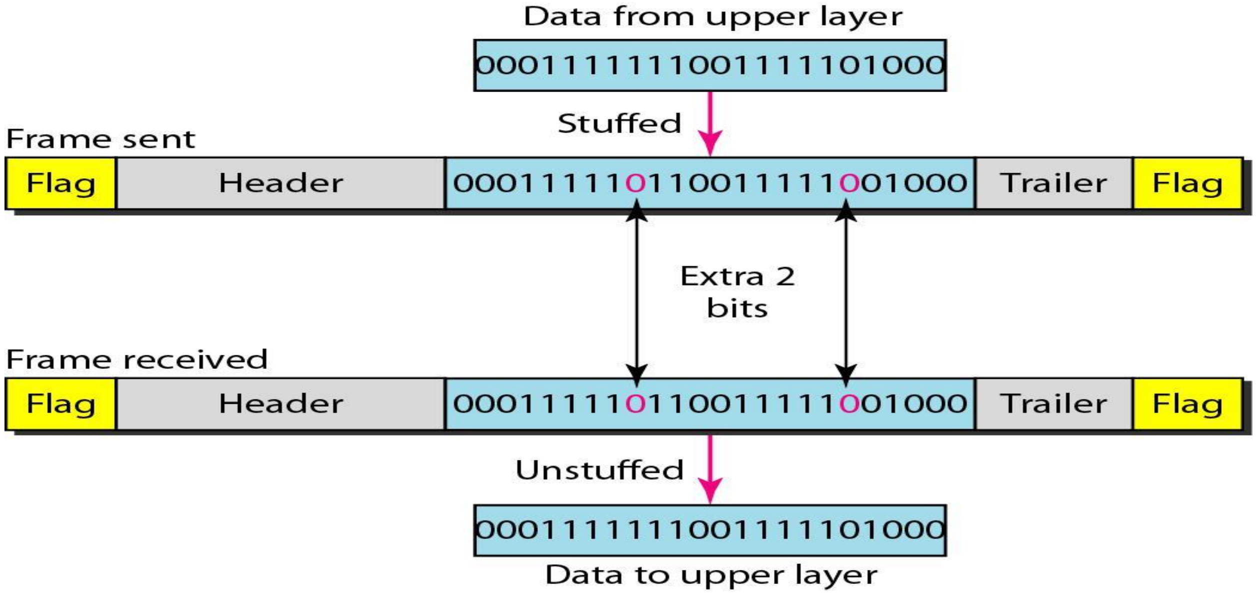


Figure (b)

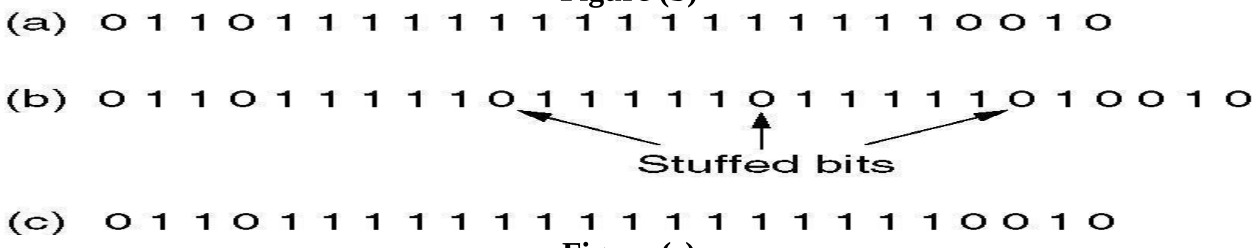

(a) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0
(b) 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0
Stuffed bits
(c) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

Figure (c)

(a) The original data.
(b) The data as they appear on the line.
(c) The data as they are stored in receiver's memory after destuffing.

Physical layer coding violation:

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 a 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.

(2) **ERROR CONTROL**

- How do we make sure that all frames are eventually delivered to the network layer at the destination and in the proper order?
- Provide sender with some acknowledgement about what is happening with the receiver
- Sender could wait for acknowledgement

Disadvantages

- If a frame vanishes, the receiver will not send an acknowledgement thus, sender will wait forever
- Dealt with by timers and sequence numbers – important part of DLL
- Sender transmits a frame, starts a timer.
- Timer set to expire after interval long enough for frame to reach destination, be processed, and have acknowledgement sent to sender
- Is a danger of frame being transmitted several times, however dealt with by assigning sequence numbers to outgoing frames, so that receiver can distinguish retransmissions from originals.

(3) **FLOW CONTROL**

What do we do when a sender transmits frames faster than the receiver can accept them?

- **Feedback-based flow control** – receiver sends back information to the sender, giving it permission to send more data or at least telling the sender how the receiver is doing
- **Rate-based flow control** – the protocol has a built-in mechanism that limits the rate at which the sender may transmit data, using feedback from the receiver.

- Because of Attenuation, distortion, noise and interferences, errors during transmission are inevitable, leading to corruption transmitted bits.

- Longer the frame size and higher the probability of single bit error, lower is the probability receiving a frame without error.

ERROR

- When data is being transmitted from one machine to another, it may possible that data become corrupted on its way. Some of the bits may be altered, damaged or lost during transmission. Such a condition is known as **error**.

TYPES OF ERRORS

- **Single bit error**: Only one bit gets corrupted. Common in Parallel transmission.

- **Burst error:** More than one bit gets corrupted very common in serial transmission of data occurs when the duration of noise is longer than the duration of one bit.

Single bit error:

- The term single-bit error means that only one bit of given data unit (such as a byte, character, or data unit) is changed from 1 to 0 or from 0 to 1 as shown in Fig. 3.2.1.
- Single bit errors are least likely type of errors in serial data transmission.
- For example, if 16 wires are used to send all 16 bits of a word at the same time and one of the wires is noisy, one bit is corrupted in each word.

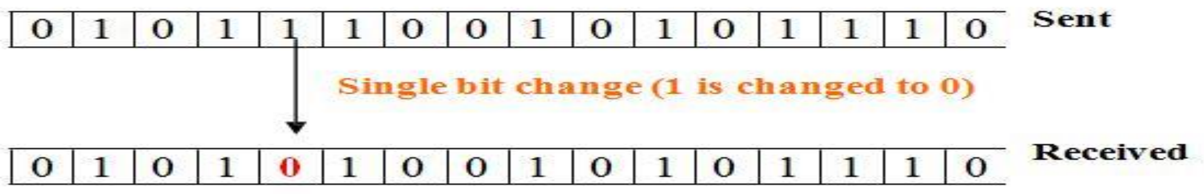


Figure 3.2.1 Single bit error

Burst error:

- More than one bit gets corrupted very common in serial transmission of data occurs when the duration of noise is longer than the duration of one bit.
- The noise affects data; it affects a set of bits.
- The number of bits affected depends on the data rate and duration of noise.

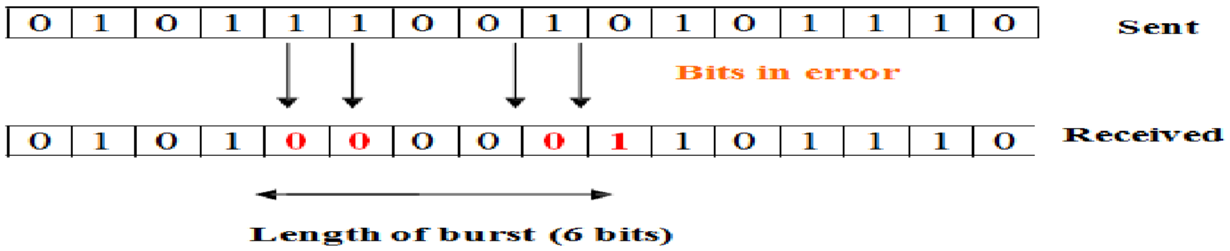


Figure 3.2.2 Burst Error

ERROR DETECTION TECHNIQUES

Basic approach used for error detection is the use of redundancy, where additional bits are added to facilitate detection and correction of errors. Popular techniques are

- Simple Parity check
- Two-dimensional Parity check
- Checksum
- Cyclic redundancy check

Redundancy is the method in which some extra bits are added to the data so as to check whether the data contain error or not.

m - data bits (i.e., message bits)
r - redundant bits (or check bits).
n - total number of bits
n= (m + r).

An n-bit unit containing data and check-bits is often referred to as an n-bit codeword.

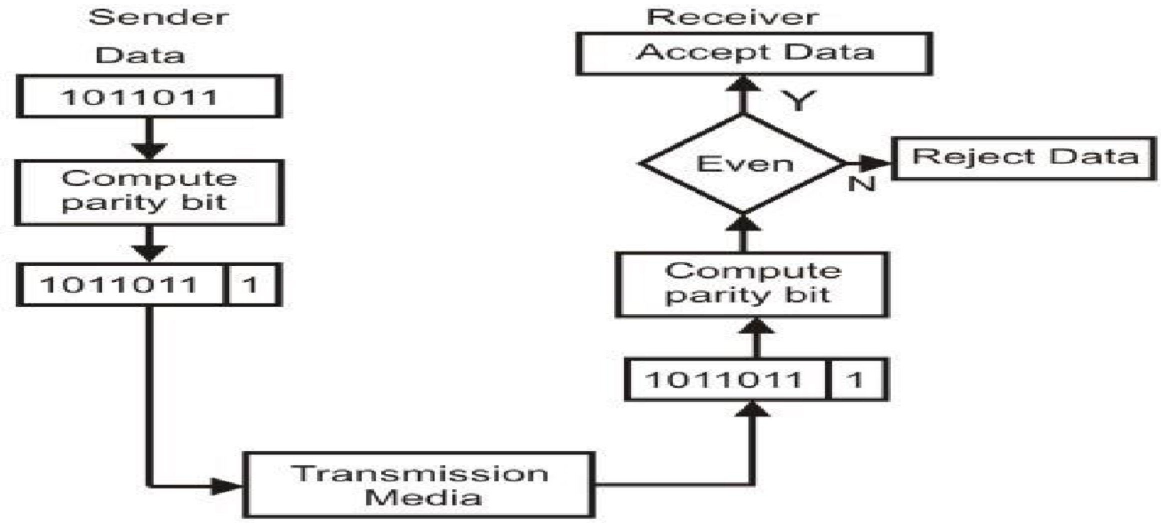
SIMPLE PARITY CHECK

The simplest and most popular error detection scheme. Appends a Parity bit to the end of the data.

Even Parity: **01000001** – Number of ones in the group of bits is even
Odd Parity: **11000001 -** Number of ones in the group of bits is odd

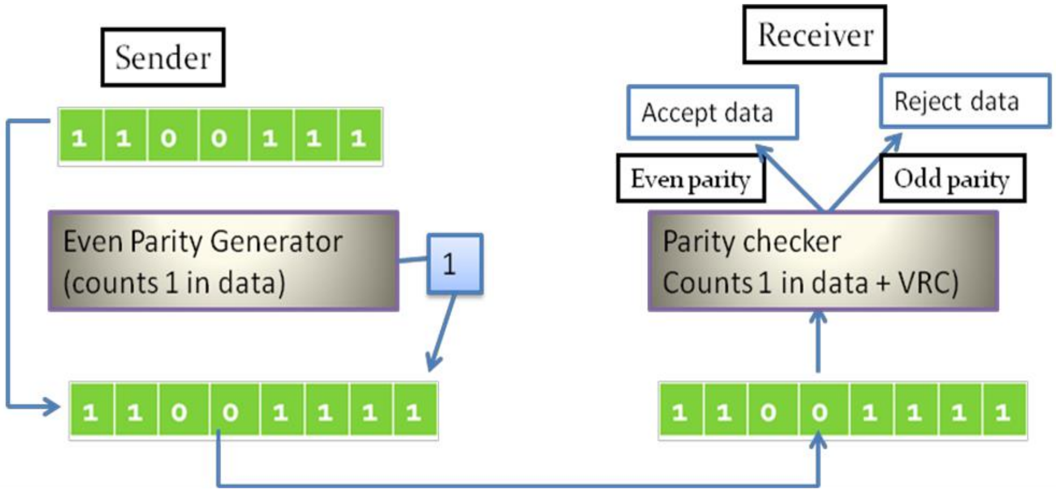
A parity of 1 is added to the block if it contains an odd number of 1's (ON bits) and 0 is added if it contains an even number of 1's. At the receiving end the parity bit is computed from the received data bits and compared with the received parity bit.

This scheme makes the total number of 1's even, that is why it is called *even parity checking*. Considering a 4-bit word, different combinations of the data words and the corresponding code words are given in Table 3.2.1.



Decimal value	Data Block	Parity bit	Code word
0	0000	0	00000
1	0001	1	00011
2	0010	1	00101
3	0011	0	00110
4	0100	1	01001
5	0101	0	01010
6	0110	0	01100
7	0111	1	01111
8	1000	1	10001
9	1001	0	10010
10	1010	0	10100
11	1011	1	10111
12	1100	0	11000
13	1101	1	11011
14	1110	1	11101
15	1111	0	11110

Example:

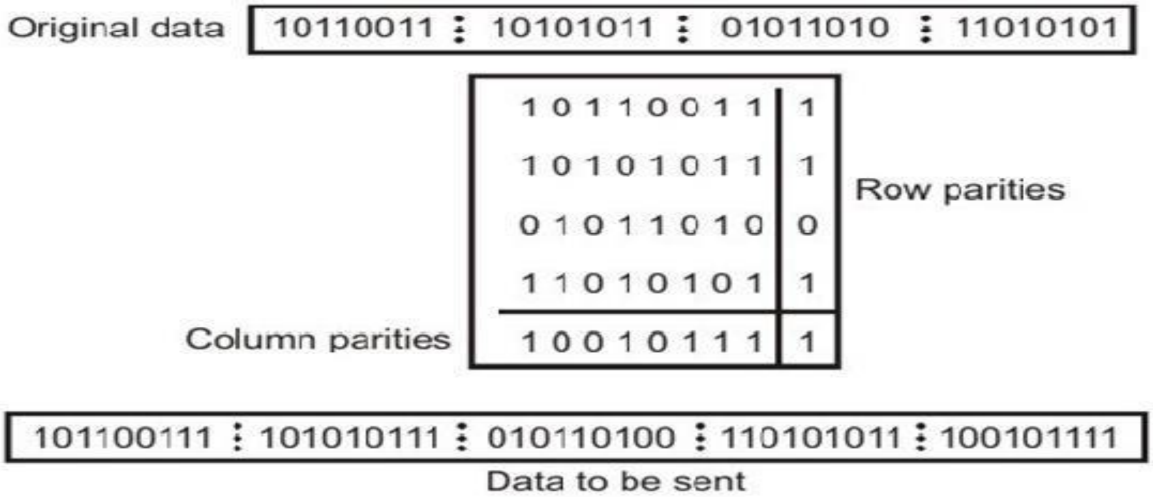


PERFORMANCE OF SIMPLE PARITY CHECK

- Simple parity check can **detect all single-bit error**
- It can also detect burst error, if the number of bits in **even or odd.**
- The technique is not foolproof against burst errors that **invert more than one bit**. If an even number of bits is inverted due to error, the **error is not detected.**

TWO-DIMENSION PARITY CHECKING

- Performance can be improved by using two dimensional parity check, which **organizes the block of bits in the form of table.**
- Parity check bits are **calculated from each row**, which is equivalent to a simple parity check.
- Parity check bits are also **calculated for all columns.**
- Both are sent along with the data.
- At the receiving end these are compared with the parity bits calculated on the received data.



Original data: 10110011 ⋮ 10101011 ⋮ 01011010 ⋮ 11010101

1 0 1 1 0 0 1 1	1
1 0 1 0 1 0 1 1	1
0 1 0 1 1 0 1 0	0
1 1 0 1 0 1 0 1	1
1 0 0 1 0 1 1 1	1

101100111 ⋮ 101010111 ⋮ 010110100 ⋮ 110101011 ⋮ 100101111

Figure 3.2.4 Two-dimension Parity Checking

Performance:

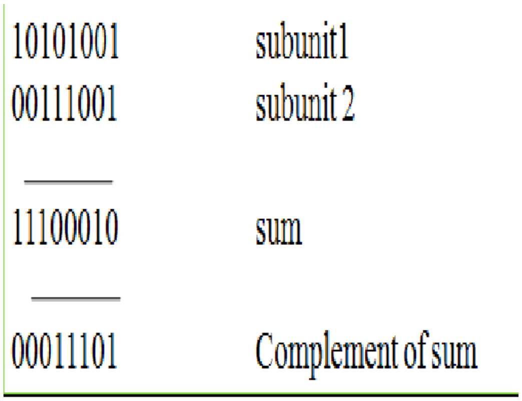
- If two bits in one data unit are damaged and two bits in exactly same position in another data unit are also damaged, The 2-D Parity check **checker will not detect an error**.
- For example, if two data units: **11001100 and 10101100.**
- If first and second from last bits in each of them is changed, making the data units as **01001110 and 00101110**, the error cannot be detected by 2-D Parity check.

CHECKSUM

- In checksum error detection scheme, the **data is divided into k segments each of m bits.**
- In the sender's end the segments are added using **1's complement arithmetic to get the sum.**
- The sum is complemented to get the checksum. The **checksum** segment is sent **along with the data segments**

Example 1:

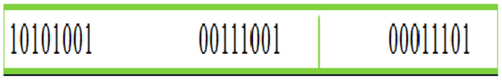
Sender **Reciever:**



10101001	subunit1
00111001	subunit 2
11100010	sum
00011101	Complement of sum

10101001	subunit1
00111001	subunit2
00011101	Checksum
11111111	sum
00000000	complement

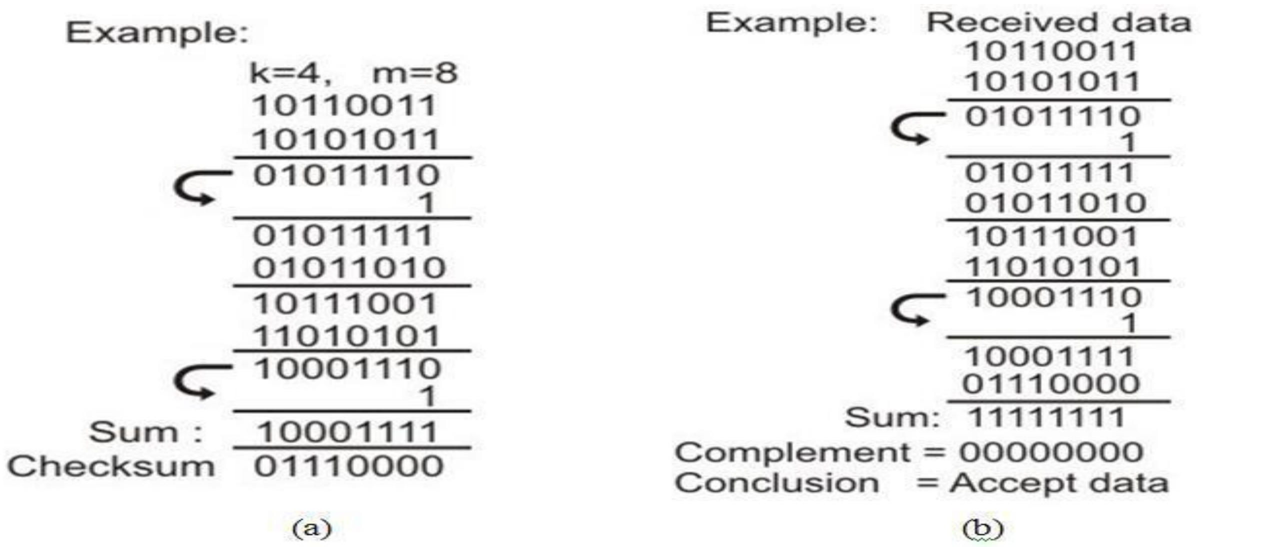
Conclusion = Accept data.



10101001 00111001 00011101

Data checksum

Example 2: K= 10110011, 10101011, 01011111, 11010101



```
Example:
            k=4,   m=8
             10110011
             10101011
            ---------
             01011110
                    1
            ---------
             01011111
             01011010
            ---------
             10111001
             11010101
            ---------
             10001110
                    1
            ---------
  Sum :      10001111
            ---------
  Checksum   01110000

                (a)
```

```
Example:   Received data
             10110011
             10101011
            ---------
             01011110
                    1
            ---------
             01011111
             01011010
            ---------
             10111001
             11010101
            ---------
             10001110
                    1
            ---------
             10001111
             01110000
            ---------
      Sum:   11111111
Complement = 00000000
Conclusion   = Accept data

                (b)
```

Figure 3.2.5 (a) Sender's end for the calculation of the checksum, (b) Receiving end for checking the checksum

CYCLIC REDUNDANCY CHECK

- One of the most powerful and commonly used error detecting codes.

Basic approach:

- Given a m-bit block of bit sequence, the sender generates an n-bit sequence known as **frame sequence check(FCS),** so that the resulting frame, consisting of m+n bits exactly divisible by **same predetermined number.**

- The receiver divides the incoming frame by that number and, if there is **no reminder, assumes there was no error.**

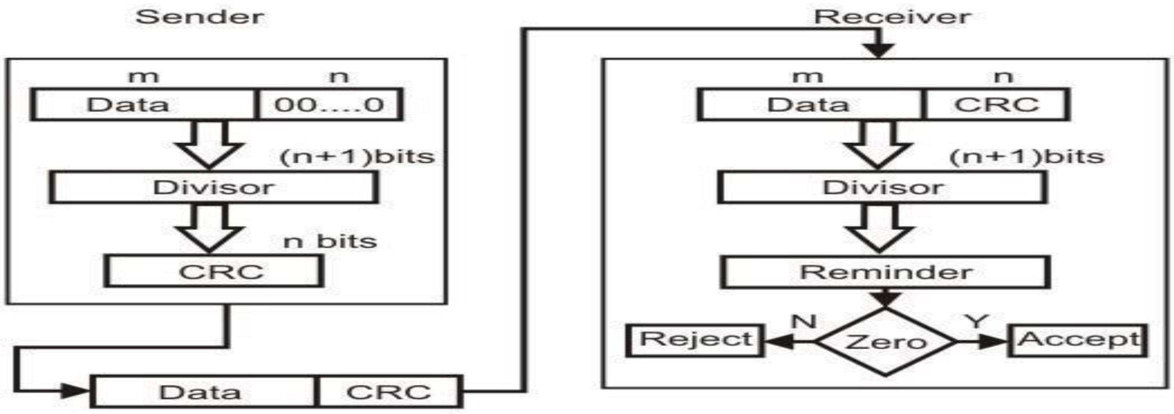


Fig. 3.2.7 by dividing a sample 4- bit number by the coefficient of the generator polynomial x^3 +x+1, which is 1011, using the modulo-2 arithmetic.

Modulo-2 arithmetic is a binary addition process without any carry over, which is just the Exclusive-OR operation.

Consider the case where k=1101. Hence we have to divide 1101000 (i.e. *k* appended by 3 zeros) by 1011, which produces the remainder r=001, so that the bit frame *(k+r)* =1101001 is actually being transmitted through the communication channel.

At the receiving end, if the received number, i.e.,1101001 is divided by the same generator polynomial 1011 to get the remainder as 000, it can be assumed that the data is free of errors.

Sender: Sender transmit the data along with remainder (**CRC**)

```
        1111  Quotient                Data: 1101
1011  1101000  <------  K           Divisor: 1011
      1011
        1100
        1011
          1110
          1011
            1010                Data to be sent:   1 1 0 1 0 0 1
            1011                                       Data | CRC
             001   Reminder (r)
```

Receiver:

```
         1111    Quotient              Data: 1101001
       ______________
  1011 | 1101001                       Divisor: 1011
         1011
        ------
          1100
          1011
         ------
           1110
           1011
          ------
             1011
             1011
          ------
            000    Reminder
```

Note: Remainder is zero, no error. Receiver can accept the data.

Performance of CRC

- CRC can detect all single-bit errors.
- CRC can detect all double-bit errors(three1's)
- CRC can detect any odd number of errors of less than the degree of the polynomial.
- CRC detects most of the larger burst errors with a high probability.

ERROR CORRECTING CODES

Concept of error-correction can be easily understood by examining the simplest case of single-bit errors. As we have already seen that a single-bit error can be detected by addition of a parity bit with the data, which needed to be send.

A single additional bit can detect error, but it's not sufficient enough to correct that error too. For correcting an error one has to know the exact position of error, i.e. exactly which bit is in error (to locate the invalid bits).

For example, to correct a single-bit error in an ASCII character, the error correction must determine which one of the seven bits is in error. To this, we have to add some additional redundant bits.

To calculate the numbers of redundant bits (r) required to correct d data bits, let us find out the relationship between the two. So we have (d+r) as the total number of bits, which are to be transmitted; then r must be able to indicate at least d+r+1 different values. Of these, one value means no error, and remaining d+r values indicate error location of error in each of d+r locations. So, d+r+1 states must be distinguishable by r bits, and r bits can indicates 2^r states. Hence, 2^r must be greater than d+r+1.

$$2^{r} >= d+r+1$$

The value of r must be determined by putting in the value of d in the relation. For example, if d is 7, then the smallest value of r that satisfies the above relation is 4. So the total bits, which are to be transmitted is 11 bits (d+r = 7+4 =11).

Now let us examine how we can manipulate these bits to discover which bit is in error. A technique developed by R.W. Hamming provides a practical solution. The solution or coding scheme he developed is commonly known as **Hamming Code**.

Hamming code can be applied to data units of any length and uses the relationship between the data bits and redundant bits as discussed.

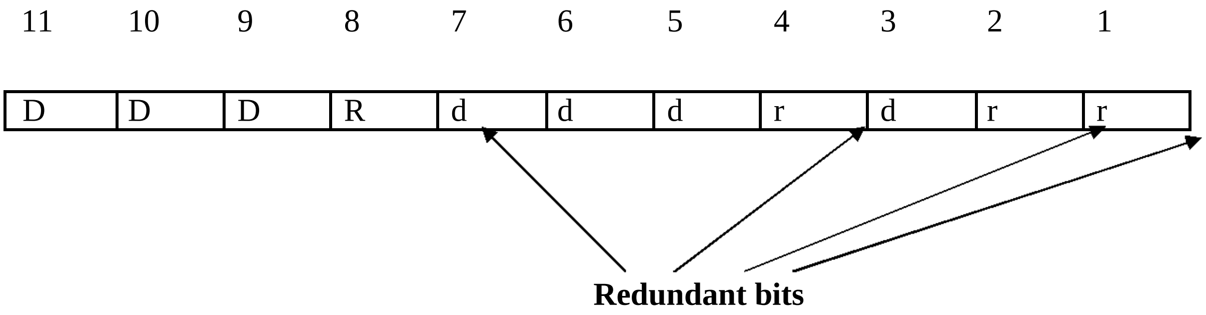
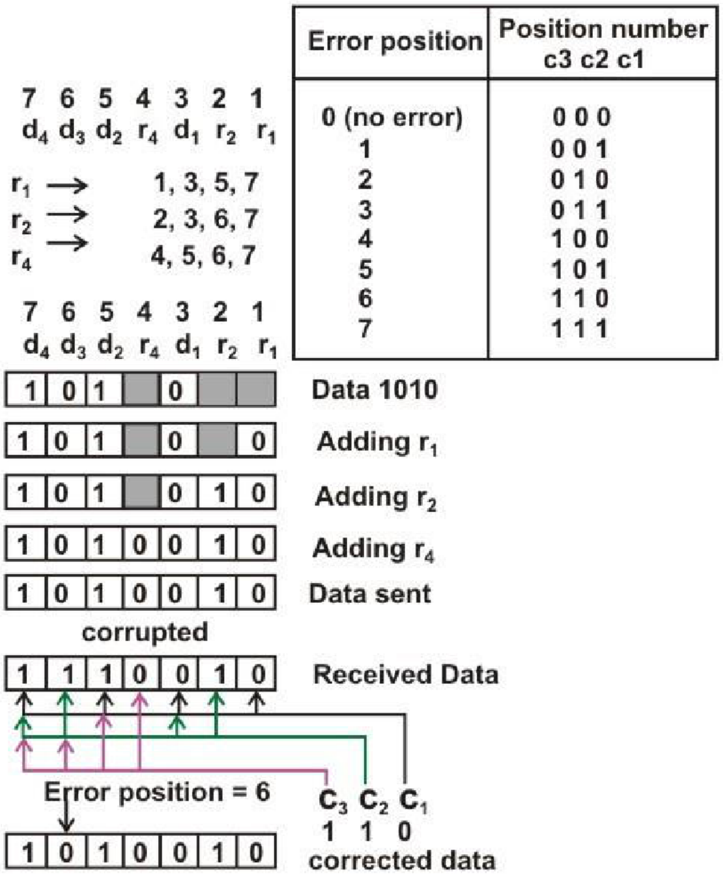


Figure 3.2.8 Positions of redundancy bits in hamming code

Basic approach for error detection by using Hamming code is as follows:

- To each group of m information bits k parity bits are added to form (m+k) bit code as shown in Fig. 3.2.8.
- Location of each of the (m+k) digits is assigned a decimal value.
- The k parity bits are placed in positions 1, 2, …, 2^{k-1} positions.–K parity checks are performed on selected digits of each codeword.
- At the receiving end the parity bits are recalculated. The decimal value of the k parity bits provides the bit-position in error, if any.



Error position	Position number c3 c2 c1
0 (no error)	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

Figure 3.2.9 Use of Hamming code for error correction for a 4-bit data

Figure 3.2.9 shows how hamming code is used for correction for 4-bit numbers (d4d3d2d1) with the help of three redundant bits (r3r2r1).

For the example data 1010, first r1 (0) is calculated considering the parity of the bit positions, 1, 3, 5 and 7. Then the parity bits r2 is calculated considering bit positions 2, 3, 6 and 7.

Finally, the parity bits r4 is calculated considering bit positions 4, 5, 6 and 7 as shown. If any corruption occurs in any of the transmitted code 1010010, the bit position in error can be found out by calculating r3r2r1 at the receiving end.

For example, if the received code word is 1110010, the recalculated value of r3r2r1 is 110, which indicates that bit position in error is 6, the decimal value of 110.

DATA LINK LAYER PROTOCOLS

i) AUTOPIAN SIMPLEX PROTOCOL

The following assumption has been made for developing the (algorithm) simplex protocol.

- The channel is a perfect noiseless channel.
- Hence an ideal channel in which no frames are lost, duplicated, or corrupted.
- No flow control and error control used.
- It is a unidirectional protocol in which data frames are traveling in only one direction- from the sender to receiver.
- Both transmitting and receiving network layer are always ready.
- Processing time that is small enough to be negligible.
- Infinite buffer space is available.

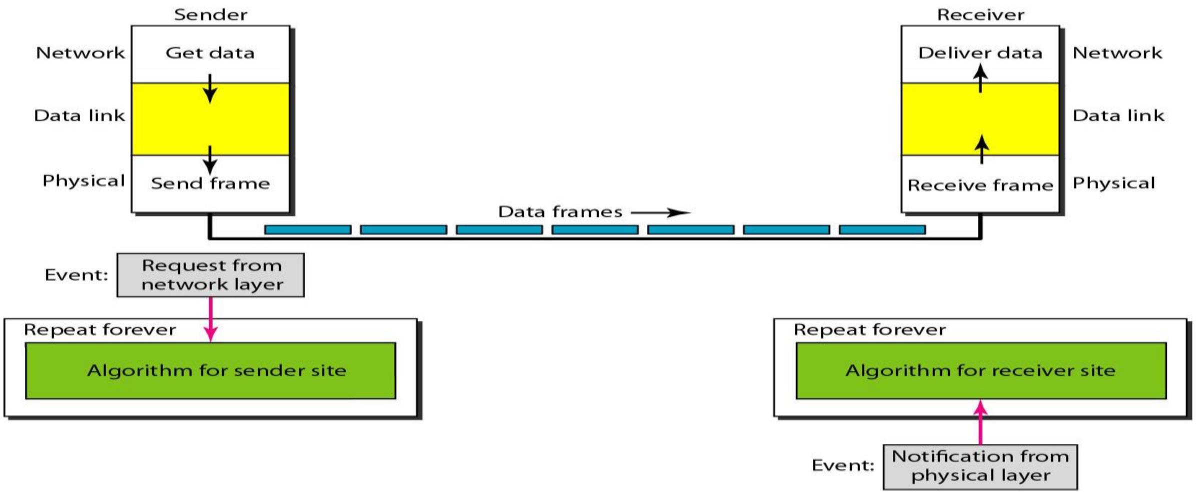


Figure3.1 shows The design of the simplest protocol with no flow or error control

Example for simplex protocol

- **Figure 3.2** below 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.

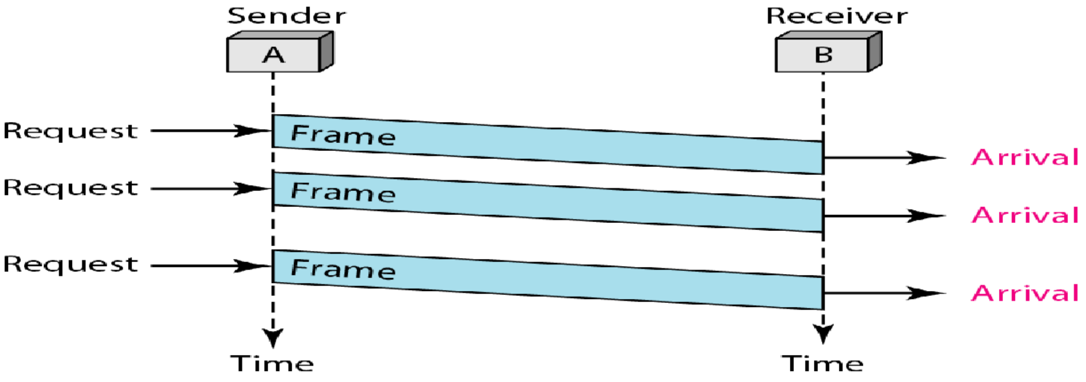


Figure3.2 Example for simplex protocol

Efficiency analysis

- Transmission in one direction
- The receiver is always ready to receive the next frame (has infinite buffer storage).
- Error-free communication channel.
- No acknowledgments or retransmissions used.
- If frame has d data bits and h overhead bits, channel bandwidth b bits/second:

Maximum Channel Utilization = data size / frame size = $d / (d + h)$
Maximum Data Throughput = $d / (d + h)$ * channel bandwidth = $d / (d + h) * b$

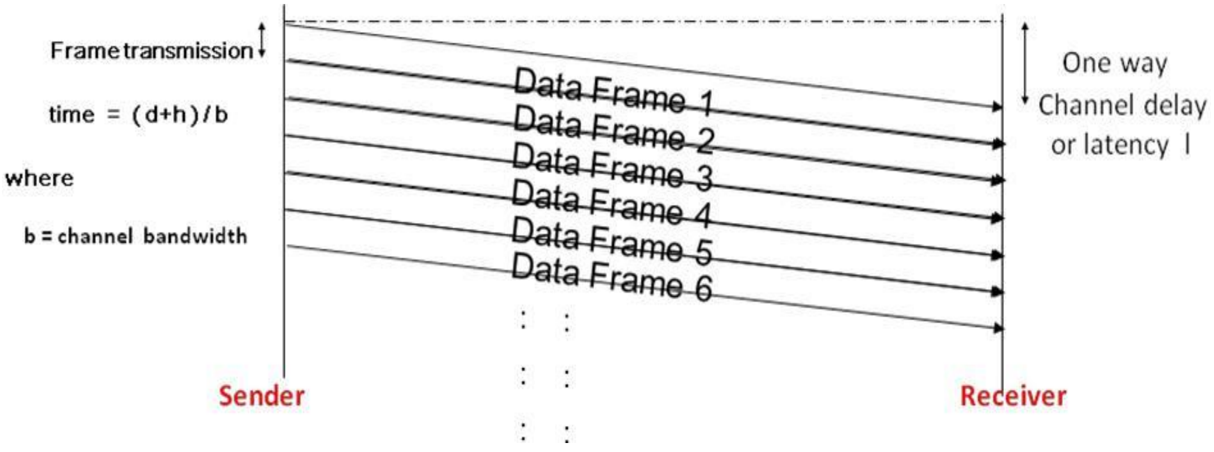


Figure 3.3. Efficiency analysis

/* Protocol 1 (Utopia) provides for data transmission in one direction only, from sender to receiver. The communication channel is assumed to be error free and the receiver is assumed to be able to process all the input infinitely quickly.Consequently, the sender just sits in a loop pumping data out onto the line as fast as it can. */

```
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 */
                   }
}

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 */
                   }
 }
```

ii) A SIMPLEX STOP-AND-WAIT PROTOCOL FOR AN ERROR-FREE CHANNEL

The following assumption has been made for developing the Stop-and-Wait Protocol

- The channel is a perfect noiseless channel.
- Flow control used
- It is a bidirectional protocol in which frames are traveling in both direction
- Both transmitting and receiving network layer are always not ready.
- Processing time considerable
- Finite buffer space is available
- The receiver may not be always ready to receive the next frame (finite buffer storage).
- Receiver sends a positive acknowledgment frame to sender to transmit the next data frame which showed in the below figure(3.4).
- Error-free communication channel assumed. No retransmissions used.
- Maximum channel utilization » (time to transmit frame /round trip time)*$d / (d + h)$
  » $d / (b * R)$
- Maximum data throughput » Channel Utilization * Channel Bandwidth
  » $d / (b * R) * b = d / R$

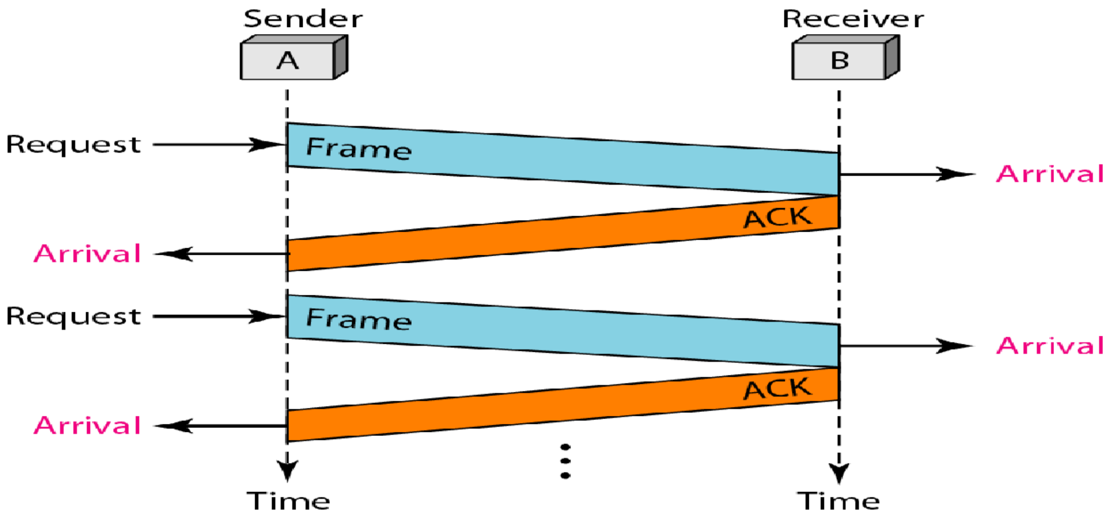


Figure 3.4. Stop-and-Wait protocol flow diagram

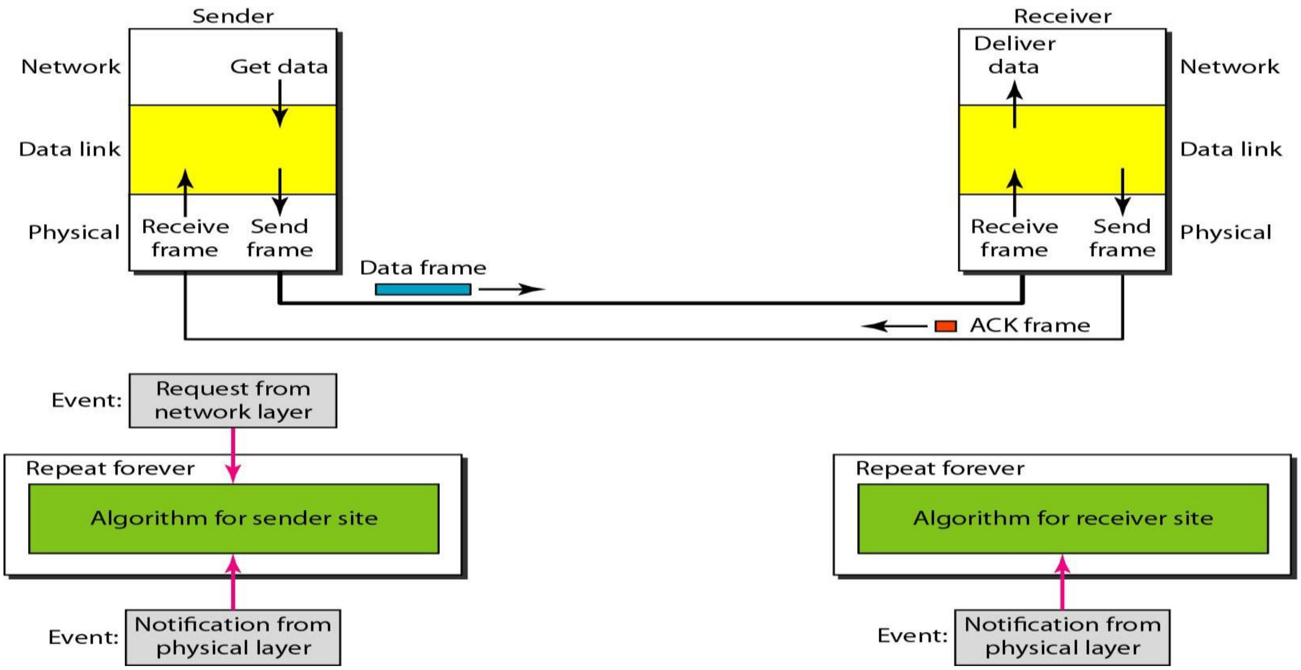


Figure 3.5 Design of Stop-and-Wait Protocol

/* Protocol 2 (Stop-and-wait) also provides for a one-directional flow of data from sender to receiver. The communication channel is once again assumed to be error free, as in protocol 1. However, this time the receiver has only a finite buffer capacity and a finite processing speed, so the protocol must explicitly prevent the sender from flooding the receiver with data faster than it can be handled. */

```
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 */
                }
}
```

iii) **A SIMPLEX STOP-AND-WAIT PROTOCOL FOR A NOISY CHANNEL**

Automatic Repeat Request (ARQ)

Purpose: To ensure a sequence of information packets is delivered in order and without errors or duplications despite transmission errors & losses.

1. <u>STOP AND WAIT WITH ARQ</u>

Automatic Repeat Request (ARQ), an error control method, is incorporated with stop and wait flow control protocol

- If error is detected by receiver, it discards the frame and send a negative ACK (NAK), causing sender to re-send the frame.

- In case a frame never got to receiver, sender has a timer: each time a frame is sent, timer is set ! If no ACK or NAK is received during timeout period, it re-sends the frame

- Timer introduces a problem: Suppose timeout and sender retransmits a frame but receiver actually received the previous transmission ! receiver has duplicated copies.

- To avoid receiving and accepting two copies of same frame, frames and ACKs are alternatively labeled 0 or 1: ACK0 for frame 1, ACK1 for frame 0.

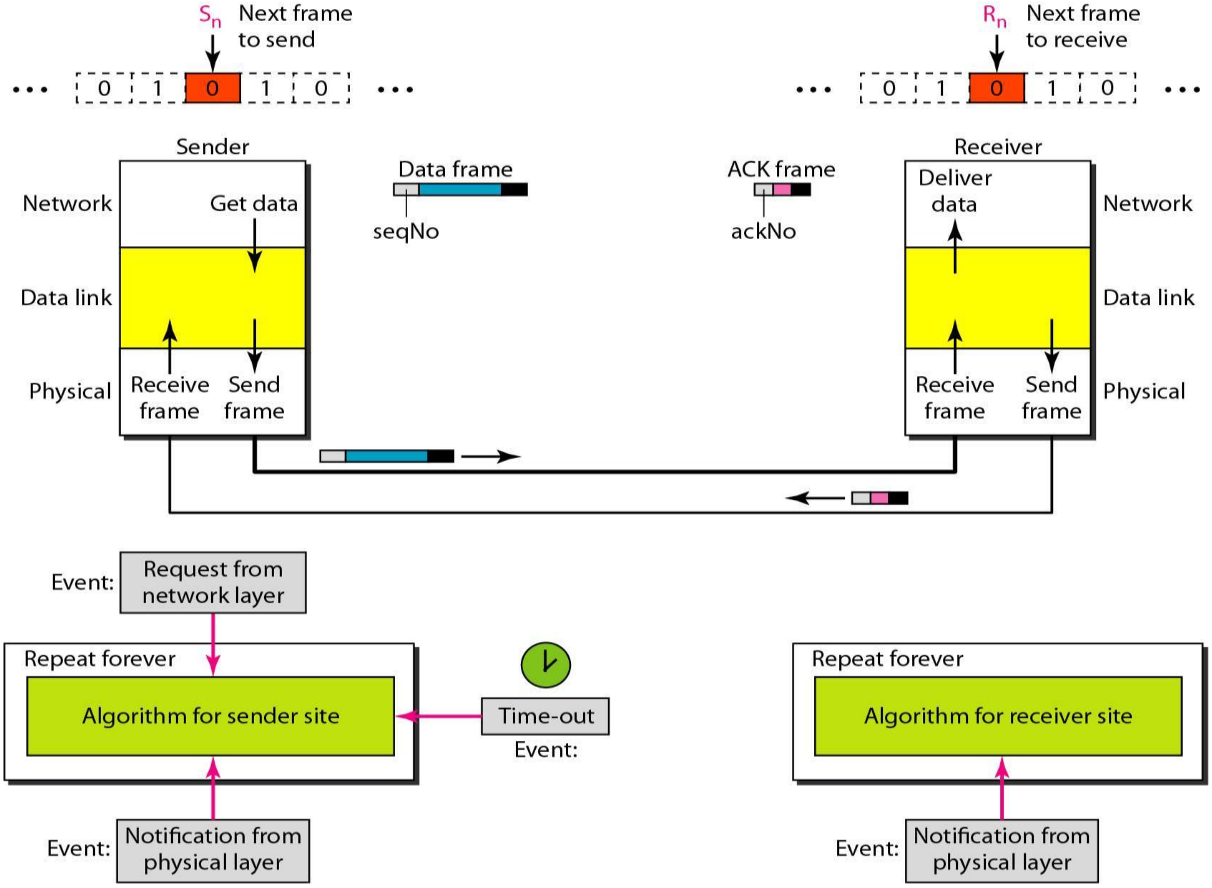


Figure 3.6 shows Design of the Stop-and-Wait ARQ Protocol

Example

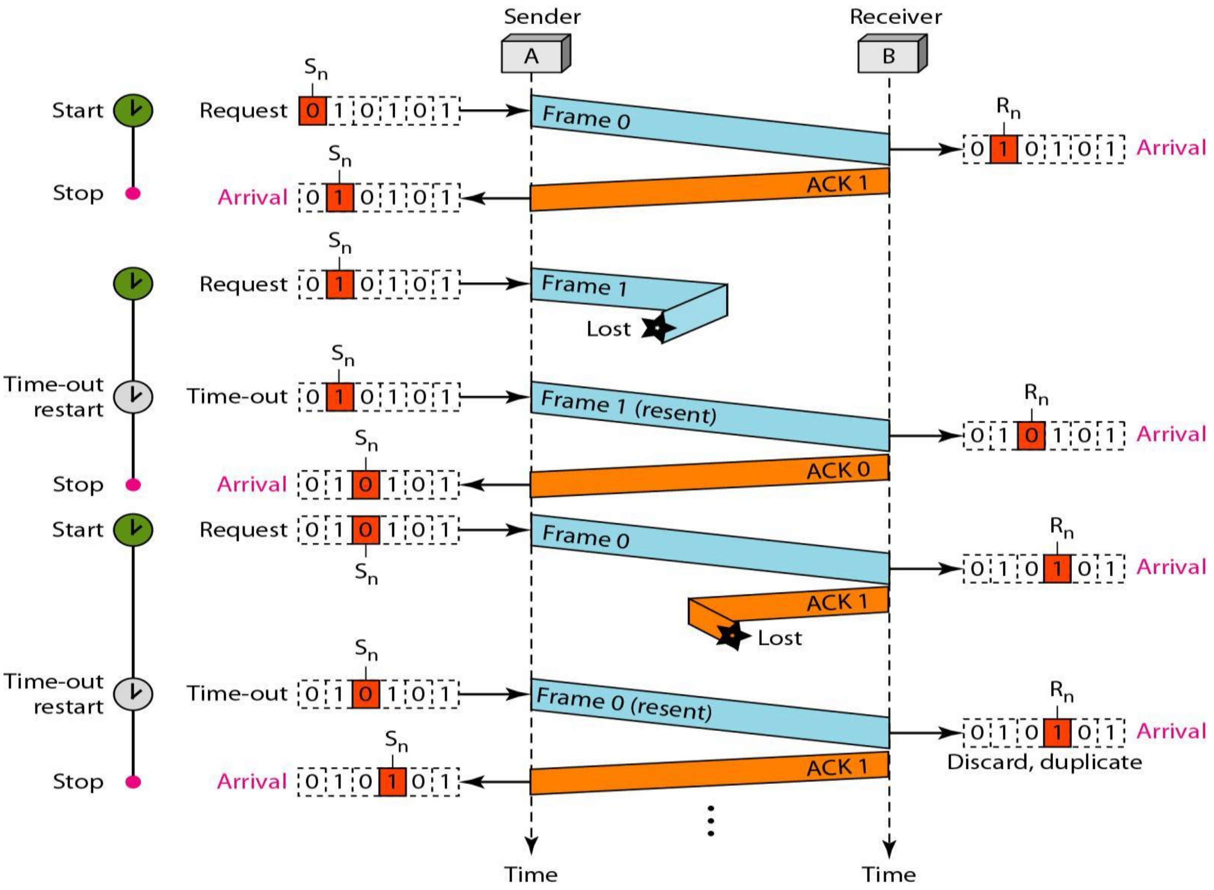


Figure 3.7 Shows an example of Stop-and-Wait ARQ.

Event :

- Frame 0 is sent and acknowledged.
- Frame 1 is lost and resent after the time-out.
- The resent frame 1 is acknowledged and the timer stops.
- Frame 0 is sent and acknowledged, but the acknowledgment is lost.
- The sender has no idea if the frame or the acknowledgment is lost.
- So after the time-out, it resends frame0, which is acknowledged.

ADVANTAGES OF STOP AND WAIT ARQ

- It can be used for noisy channels
- It has both error and flow control mechanism
- It has timer implementation

DISADVANTAGES OF STOP AND WAIT ARQ

- Efficiency is very less.
- Only 1 frame is sent at a time
- Timer should be set for each individual frame

- No pipelining
- sender window size is 1(disadvantage over Go back N ARQ)
- receiver window size is 1(disadvantage over selective repeat ARQ)

```
  /* Protocol 3 (PAR) allows unidirectional data flow over an unreliable channel. */

#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 */ }

        }
}
```

SLIDING WINDOW PROTOCOLS

1. **A One-Bit Sliding Window Protocol**
2. **A Protocol Using Go-Back-N**
3. **A Protocol Using Selective Repeat**

i) A ONE-BIT SLIDING WINDOW PROTOCOL

The starting machine fetches the first packet from its network layer, builds a frame from it, and sends it. When this (or any) 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.

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.

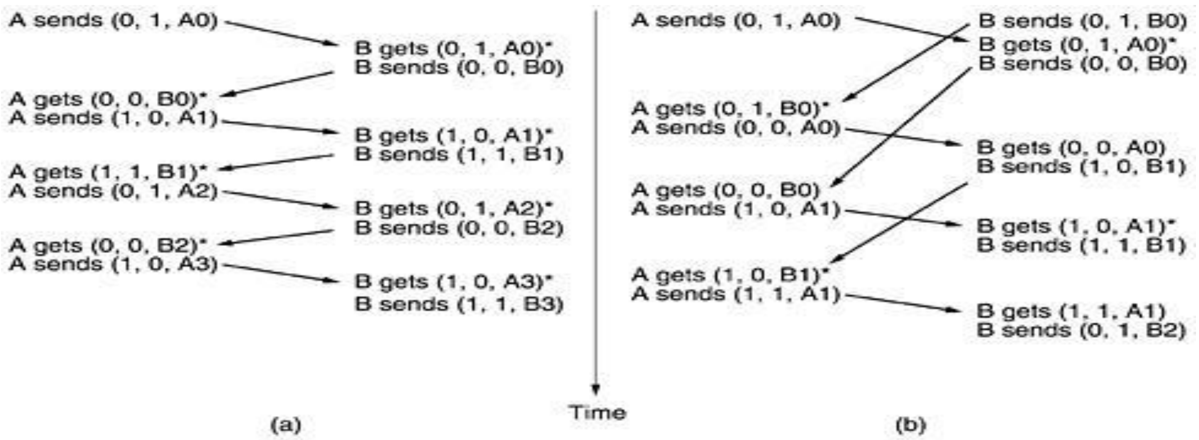
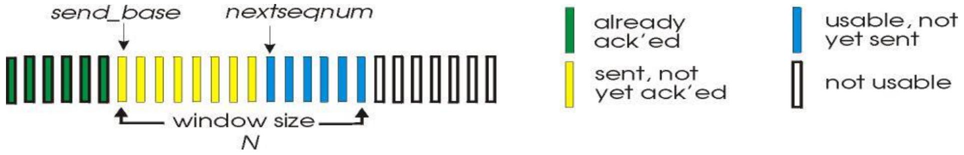


Fig 2.8: Two scenarios for protocol 4. (a) Normal case. (b) Abnormal case. The notation is (seq, ack, packet number). An asterisk indicates where a network layer accepts a packet

ii) A PROTOCOL USING GO-BACK-N

In a Go-Back-N (GBN) protocol, the sender is allowed to transmit multiple packets (when available) without waiting for an acknowledgment, but is constrained to have no more than some maximum allowable number, *N*, of unacknowledged packets in the pipeline.



Sender's view of sequence numbers in Go-Back-N

The above figure shows the sender's view of the range of sequence numbers in a GBN protocol. If we define *base* to be the sequence number of the oldest unacknowledged packet and *nextseqnum* to be the smallest unused sequence number (i.e., the sequence number of the next packet to be sent), then four intervals in the range of sequence numbers can be identified.

Sequence numbers in the interval *[0,base-1]* correspond to packets that have already been transmitted and acknowledged.

The interval *[base, nextseqnum-1]* corresponds to packets that have been sent but not yet acknowledged. Sequence numbers in the interval *[nextseqnum,base+N-1]* can be used for packets that can be sent immediately, should data arrive from the upper layer. Finally, sequence numbers greater than or equal to *base+N* can not be used until an unacknowledged packet currently in the pipeline has been acknowledged.

The range of permissible sequence numbers for transmitted but not-yet-acknowledged packets can be viewed as a ``window" of size *N* over the range of sequence numbers. *N* is often referred to as the **window size** and the GBN protocol itself as a **sliding window protocol.**

The GBN sender must respond to three types of events:

- **Invocation from above.** When rdt_send() is called from above, the sender first checks to see if the window is full, i.e., whether there are *N* outstanding, unacknowledged packets. If the window is not full, a packet is created and sent, and variables are appropriately updated. If the window is full, the sender simply returns the data back to the upper layer, an implicit indication that the window is full.
- **Receipt of an ACK.** In our GBN protocol, an acknowledgement for packet with sequence number *n* will be taken to be a **cumulative acknowledgement,** indicating that all packets with a sequence number up to and including n have been correctly received at the receiver. We'll come back to this issue shortly when we examine the receiver side of GBN.

- **A timeout event.** The protocol's name, ``Go-Back-N," is derived from the sender's behavior in the presence of lost or overly delayed packets. As in the stop-and-wait protocol, a timer will again be used to recover from lost data or acknowledgement packets. If a timeout occurs, the sender resends *all* packets that have been previously sent but that have not yet been acknowledged. If an ACK is received but there are still additional transmitted-but-yet-to-be-acknowledged packets, the timer is restarted. If there are no outstanding unacknowledged packets, the timer is stopped.

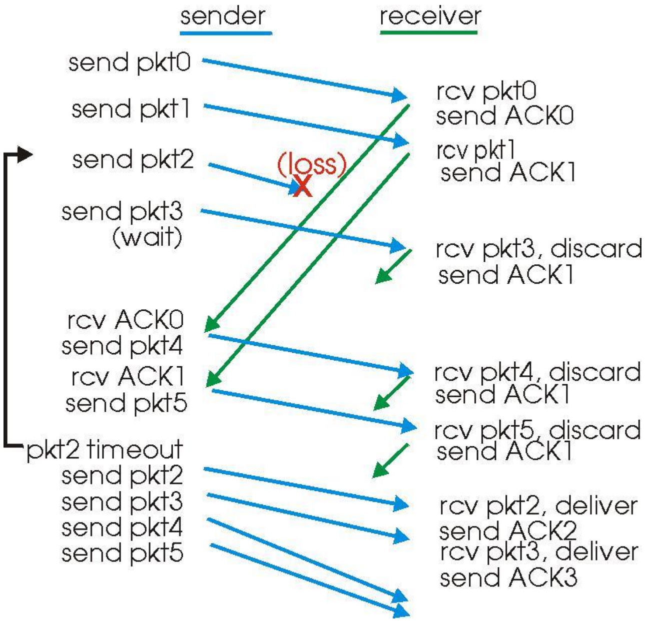
The receiver's actions in GBN are also simple. If a packet with sequence number *n* is received correctly and is in-order (i.e., the data last delivered to the upper layer came from a packet with sequence number *n-1*), the receiver sends an ACK for packet *n* and delivers the data portion of the packet to the upper layer.

In all other cases, the receiver discards the packet and resends an ACK for the most recently received in-order packet. Note that since packets are delivered one-at-a-time to the upper layer, if packet *k* has been received and delivered, then all packets with a sequence number lower than *k* have also been delivered. Thus, the use of cumulative acknowledgements is a natural choice for GBN.

In our GBN protocol, the receiver discards out-of-order packets. While it may seem silly and wasteful to discard a correctly received (but out-of-order) packet, there is some justification for doing so. Recall that the receiver must deliver data, in-order, to the upper layer.

Suppose now that packet *n* is expected, but packet *n+1* arrives. Since data must be delivered in order, the receiver *could* buffer (save) packet *n+1* and then deliver this packet to the upper layer after it had later received and delivered packet *n.* However, if packet *n* is lost, both it and packet *n+1* will eventually be retransmitted as a result of the GBN retransmission rule at the sender. Thus, the receiver can simply discard packet *n+1.*

The advantage of this approach is the simplicity of receiver buffering - the receiver need not buffer *any* out-of-order packets. Thus, while the sender must maintain the upper and lower bounds of its window and the position of *nextseqnum* within this window, the only piece of information the receiver need maintain is the sequence number of the next in-order packet. Of course, the disadvantage of throwing away a correctly received packet is that the subsequent retransmission of that packet might be lost or garbled and thus even more retransmissions would be required.



Go-Back-N in operation

In the above figure, shows the operation of the GBN protocol for the case of a window size of four packets. Because of this window size limitation, the sender sends packets 0 through 3 but then must wait for one or more of these packets to be acknowledged before proceeding. As each successive ACK (e.g., ACK0 and ACK1) is received, the window slides forwards and the sender can transmit one new packet (pkt4 and pkt5, respectively).

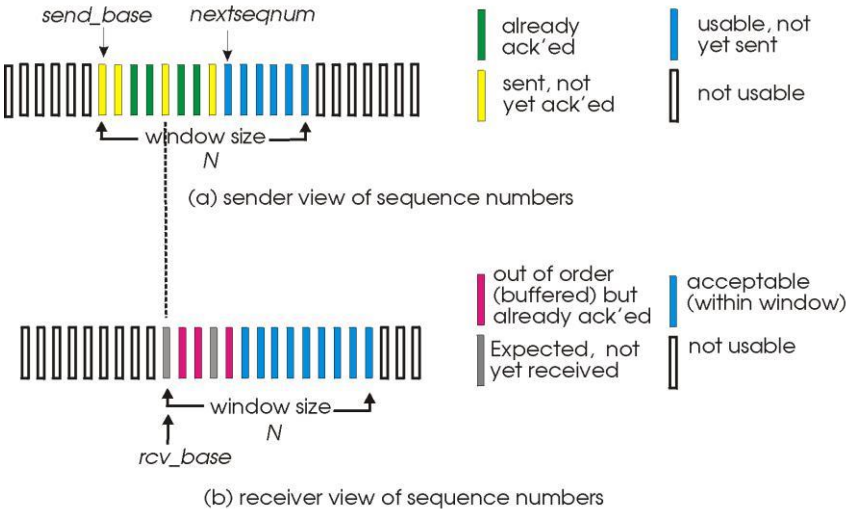
On the receiver side, packet 2 is lost and thus packets 3, 4, and 5 are found to be out-of-order and are discarded.

The implementation would also likely be in the form of various procedures that implement the actions to be taken in response to the various events that can occur. In such **event-based programming,** the various procedures are called (invoked) either by other procedures in the protocol stack, or as the result of an interrupt.

In the sender, these events would be *(i)* a call from the upper layer entity to invoke rdt_send(), *(ii)* a timer interrupt, and *(iii)* a call from the lower layer to invoke rdt_rcv() when a packet arrives. The programming exercises at the end of this chapter will give you a chance to actually implement these routines in a simulated, but realistic, network setting.

iii) A PROTOCOL USING SELECTIVE REPEAT

Selective Repeat (SR) protocols avoid unnecessary retransmissions by having the sender retransmit only those packets that it suspects were received in error (i.e., were lost or corrupted) at the receiver. This individual, as-needed, retransmission will require that the receiver *individually* acknowledge correctly-received packets. A window size of *N* will again be used to limit the number of outstanding, unacknowledged packets in the pipeline.


send_base
nextseqnum
window size
N
(a) sender view of sequence numbers
already ack'ed
sent, not yet ack'ed
usable, not yet sent
not usable
out of order (buffered) but already ack'ed
Expected, not yet received
acceptable (within window)
not usable
window size
N
rcv_base
(b) receiver view of sequence numbers

SR sender and receiver views of sequence number space

The SR receiver will acknowledge a correctly received packet whether or not it is in-order. Out-of-order packets are buffered until any missing packets (i.e., packets with lower sequence numbers) are received, at which point a batch of packets can be delivered in-order to the upper layer. Figure receiver itemizes the various actions taken by the SR receiver.

Selective Repeat sender actions

- **Data received from above.** When data is received from above, the SR sender checks the next available sequence number for the packet. If the sequence number is within the sender's window, the data is packetized and sent; otherwise it is either buffered or returned to the upper layer for later transmission, as in GBN.

- **Timeout.** Timers are again used to protect against lost packets. However, each packet must now have its own logical timer, since only a single packet will be transmitted on timeout. A single hardware timer can be used to mimic the operation of multiple logical timers.
- **ACK received.** If an ACK is received, the SR sender marks that packet as having been received, provided it is in the window. If the packet's sequence number is equal to sendbase, the window base is moved forward to the unacknowledged packet with the smallest sequence number. If the window

moves and there are untransmitted packets with sequence numbers that now fall within the window, these packets are transmitted.

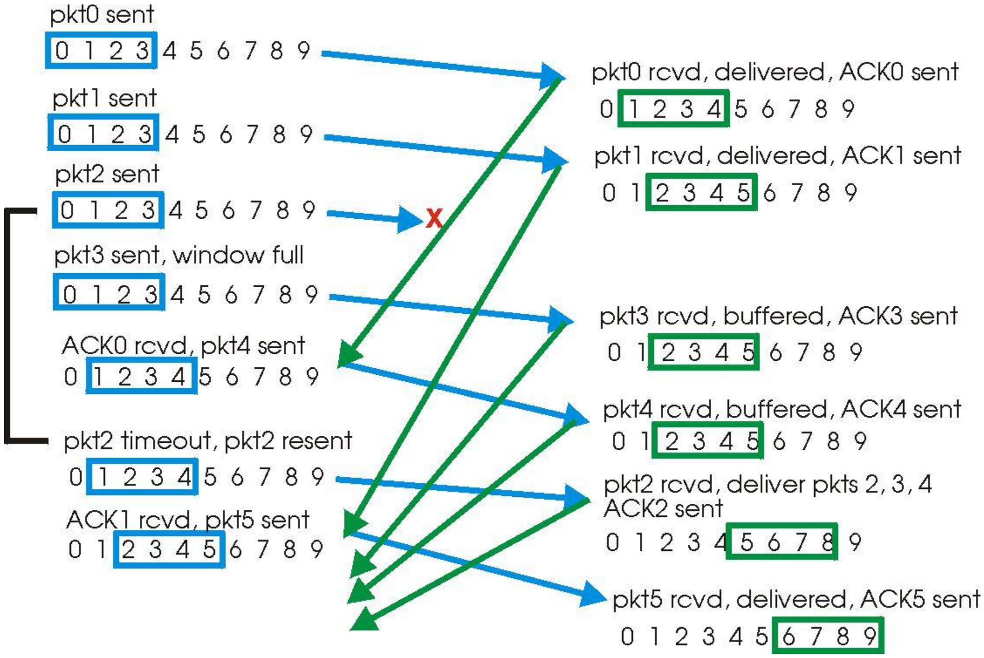
Selective Repeat Receiver Actions

- **Packet with sequence number in [*rcvbase, rcvbase+N-1*] is correctly received.** In this case, the received packet falls within the receivers window and a selective ACK packet is returned to the sender. If the packet was not previously received, it is buffered. If this packet has a sequence number equal to the base of the receive window, then this packet, and any previously buffered and consecutively numbered (beginning with *rcvbase)* packets are delivered to the upper layer. The receive window is then moved forward by the number of packets delivered to the upper layer.

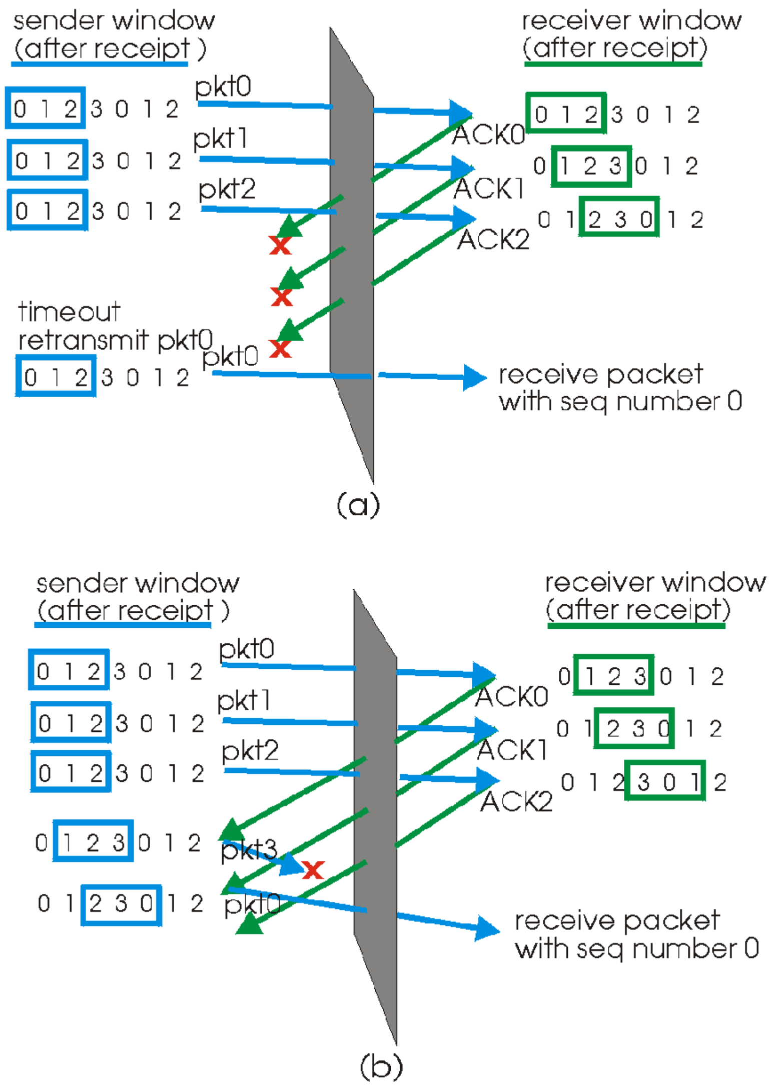
- **Packet with sequence number in *[rcvbase-N,rcvbase-1*] is received.** In this case, an ACK must be generated, even though this is a packet that the receiver has previously acknowledged.

- **Otherwise.** Ignore the packet.

The below figure shows an example of SR operation in the presence of lost packets. The receiver initially buffers packets 3 and 4, and delivers them together with packet 2 to the upper layer when packet 2 is finally received.



SR Operation



The lack of synchronization between sender and receiver windows has important consequences when we are faced with the reality of a finite range of sequence numbers.

Consider what could happen, for example, with a finite range of four packet sequence numbers, 0,1,2,3 and a window size of three. Suppose packets 0 through 2 are transmitted and correctly received and acknowledged at the receiver.

At this point, the receiver's window is over the fourth, fifth and sixth packets, which have sequence numbers 3, 0, and 1, respectively. Now consider two scenarios.

In the first scenario, shown in Figure (a), the ACKs for the first three packets are lost and the sender retransmits these packets. The receiver thus next receives a packet with sequence number 0 - a copy of the first packet sent.

In the second scenario, shown in Figure 3.4-19(b), the ACKs for the first three packets are all delivered correctly. The sender thus moves its window forward and sends the fourth, fifth and sixth packets, with sequence numbers 3, 0, 1, respectively. The packet with sequence number 3 is lost, but the packet with sequence number 0 arrives - a packet containing *new* data.

MEDIUM ACCESS CONTROL SUBLAYER

The MAC sublayer is the bottom part of the data link layer. The protocols used to determine who goes next on a multiaccess channel belong to a sublayer of the data link layer called the **MAC** (**Medium Access Control**) sublayer. The MAC sublayer is especially important in LANs, particularly wireless ones because wireless is naturally a broadcast channel. broadcast channels are sometimes referred to as **multi-access channels** or **random access channels**.

THE CHANNEL ALLOCATION PROBLEM

The channel allocation problem is how to allocate a single broadcast channel among competing users.

STATIC CHANNEL ALLOCATION IN LANs AND MANs

- The traditional way of allocating a single channel, such as a telephone trunk, among multiple competing users is **Frequency Division Multiplexing** (FDM).
- If there are *N* users, the bandwidth is divided into *N* equal-sized portions, each user being assigned one portion. Since each user has a private frequency band, there is no interference between users.
- When there is only a small and constant number of a user, each of which has a heavy (buffered) load of traffic (e.g., carriers' switching offices), FDM is a simple and efficient allocation mechanism.
- However, when the number of senders is large and continuously varying or the traffic is bursty, FDM presents some problems.
- If the spectrum is cut up into *N* regions and fewer than *N* users are currently interested in communicating, a large piece of valuable spectrum will be wasted.
- If more than *N* users want to communicate, some of them will be denied permission for lack of bandwidth, even if some of the users who have been assigned a frequency band hardly ever transmit or receive anything.
- However, even assuming that the number of users could somehow be held constant at *N*, dividing the single available channel into static sub channels is inherently inefficient.
- The basic problem is that when some users are quiescent, their bandwidth is simply lost. They are not using it, and no one else is allowed to use it either.
- Furthermore, in most computer systems, data traffic is extremely bursty (peak traffic to mean traffic ratios of 1000:1 are common). Consequently, most of the channels will be idle most of the time.
- The poor performance of static FDM can easily be seen from a simple queuing theory calculation. Let us start with the mean time delay, *T*, for a channel of capacity *C* bps, with an arrival rate of λ frames/sec, each frame having a length drawn from an exponential probability density function with mean $1/\mu$ bits/frame. With these parameters the arrival rate is λ frames/sec and the service rate is μC frames/sec. From queuing theory it can be shown that for Poisson arrival and service times,

$$T = \frac{1}{\mu C - \lambda}$$

- For example, if *C* is 100 Mbps, the mean frame length, 1/μ, is 10,000 bits, and the frame arrival rate, λ, is 5000 frames/sec, then *T* = 200 μ sec. Note that if we ignored the queuing delay and just asked how long it takes to send a 10,000 bit frame on a 100-Mbps network, we would get the (incorrect) answer of 100 μ sec. That result only holds when there is no contention for the channel.

- Now let us divide the single channel into *N* independent sub channels, each with capacity *C/N* bps. The mean input rate on each of the sub channels will now be λ*/N*. Recomputing *T* we get

Equation 4

$$T_{\text{FDM}} = \frac{1}{\mu(C/N) - (\lambda/N)} = \frac{N}{\mu C - \lambda} = NT$$

- The mean delay using FDM is *N* times worse than if all the frames were somehow magically arranged orderly in a big central queue.

- Precisely the same arguments that apply to FDM also apply to time division multiplexing (TDM). Each user is statically allocated every *N*th time slot. If a user does not use the allocated slot, it just lies fallow. The same holds if we split up the networks physically. Using our previous example again, if we were to replace the 100-Mbps network with 10 networks of 10 Mbps each and statically allocate each user to one of them, the mean delay would jump from 200 μ sec to 2 msec.

- Since none of the traditional static channel allocation methods work well with bursty traffic, we will now explore dynamic methods.

DYNAMIC CHANNEL ALLOCATION IN LANs AND MANSs

FIVE KEY ASSUMPTIONS

Station Model.

- The model consists of *N* independent **stations** (e.g., computers, telephones, or personal communicators), each with a program or user that generates frames for transmission. Stations are sometimes called **terminals**.

- The probability of a frame being generated in an interval of length *t* is λΔ*t*, where λ is a constant (the arrival rate of new frames). Once a frame has been generated, the station is blocked and does nothing until the frame has been successfully transmitted.

Single Channel Assumption.

- A single channel is available for all communication. All stations can transmit on it and all can receive from it.

- As far as the hardware is concerned, all stations are equivalent, although protocol software may assign priorities to them.

Collision Assumption.

- If two frames are transmitted simultaneously, they overlap in time and the resulting signal is garbled. This event is called a **collision**.

- All stations can detect collisions. A collided frame must be transmitted again later. There are no errors other than those generated by collisions.

4a. Continuous Time.

Frame transmission can begin at any instant. There is no master clock dividing time into discrete intervals.

4b. Slotted Time.

- Time is divided into discrete intervals (slots). Frame transmissions always begin at the start of a slot.
- A slot may contain 0, 1, or more frames, corresponding to an idle slot, a successful transmission, or a collision, respectively.

5a. Carrier Sense.

Stations can tell if the channel is in use before trying to use it. If the channel is sensed as busy, no station will attempt to use it until it goes idle.

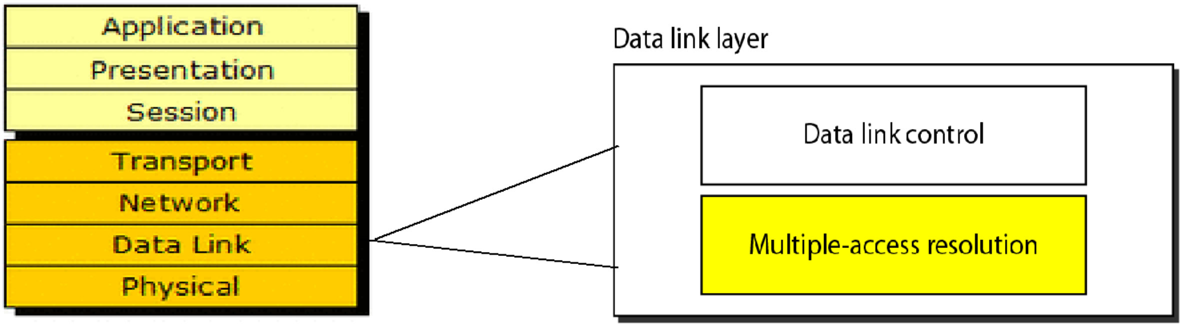
5b. No Carrier Sense.

- Stations cannot sense the channel before trying to use it. They just go ahead and transmit. Only later can they determine whether the transmission was successful.

MULTIPLE ACCESS PROTOCOL

INTRODUCTION

- The media access control (MAC) data communication protocol sub-layer, also known as the **medium access control**, is a **sublayer of the data link** layer specified in the seven-layer OSI model.
- It provides addressing and channel access control mechanisms that make it possible for several terminals or network nodes to communicate within a multiple access network that incorporates a shared medium, e.g. Ethernet. The hardware that implements the MAC is referred to as a medium access controller.



□ The MAC sub-layer acts as an interface between the logical link control (LLC) sublayer and the network's physical layer. The MAC layer emulates a full-duplex logical communication channel in a multi-point network. This channel may provide unicast, multicast or broadcast communication service.

- The channel access control mechanisms provided by the MAC layer is known as a **multiple access protocol**. This makes it possible for several stations connected to the same physical medium to share it.

- Examples of shared physical media are bus networks, ring networks, hub networks, wireless networks and half-duplex point-to-point links.

- The multiple access protocol may detect or avoid data packet collisions if a packet mode contention based channel access method is used, or reserve resources to establish a logical channel if a circuit switched or channelization based channel access method is used. The channel access control mechanism relies on a physical layer multiplex scheme.

I. RANDOM ACCESS PROTOCOLS

- In a random access protocol, a transmitting node always transmits at the full rate of the channel, namely, R bps.

- When there is a collision, each node involved in the collision repeatedly retransmits its frame(that is ,packet) until the frame gets through without a collision. But when a node experiences a collision, it doesn't necessarily retransmitting the frame right away. Instead it waits a random delay before retransmitting the frame.

- Each node involved in a collision chooses independent random delays .Because the random delays are independently chosen, it is possible that one of the nodes will pick a delay that is sufficiently less than the delays of the other colliding nodes and will therefore be able to sneak its frame into the channel without a collision.

The most commonly used random access protocols

1. The ALOHA protocol ,
2. CSMA (carrier sense multiple access) protocol ,
3. CSMA/CD (carrier sense multiple access /collision detection) protocol and
4. Collision-Free Protocols
5. Limited-Contention Protocols

1. ALOHA

- In the 1970s, Norman Abramson and his colleagues at the University of Hawaii devised a new and elegant **method to solve the channel allocation problem**.
- Although Abramson's work, called the ALOHA system, used ground-based radio broadcasting, the basic idea is applicable to any system in which uncoordinated users are competing for the use of a single shared channel.
- The two versions of ALOHA here: **pure and slotted**.
- They differ with respect to whether time is divided into discrete slots into which all frames must fit.
- Pure ALOHA does not require global time synchronization; slotted ALOHA does.

PURE ALOHA

- If you have data to send, send the data
- If the message collides with another transmission, try resending "later"

Note that the first step implies that Pure ALOHA does not check whether the channel is busy before transmitting. The critical aspect is the "later" concept: the quality of the backoff scheme chosen significantly influences the efficiency of the protocol, the ultimate channel capacity, and the predictability of its behavior.

A sketch of frame generation in an ALOHA system is given in Fig. 4-1. We have made the frames all the same length because the throughput of ALOHA systems is maximized by having a uniform frame size rather than by allowing variable length frames.

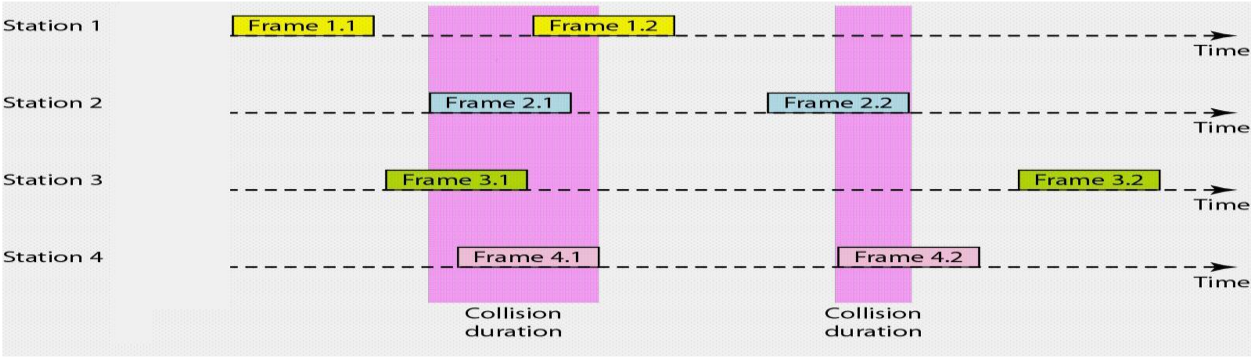
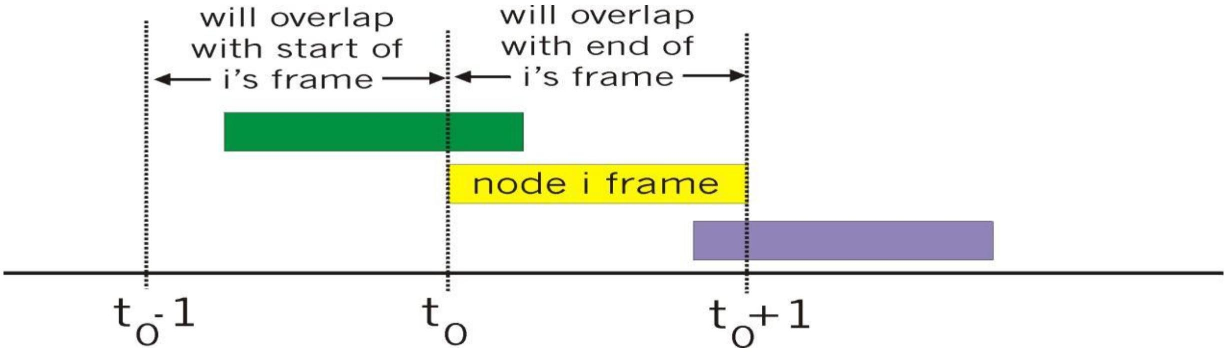


Figure 4-1. In pure ALOHA, frames are transmitted at completely arbitrary times.

To assess Pure ALOHA, we need to predict its throughput, the rate of (successful) transmission of frames. First, let's make a few simplifying assumptions:

- All frames have the same length.
- Stations cannot generate a frame while transmitting or trying to transmit.
- The population of stations attempts to transmit (both new frames and old frames that collided) according to a Poisson distribution.

Let "*T*" refer to the time needed to transmit one frame on the channel, and let's define "frame-time" as a unit of time equal to *T*. Let "*G*" refer to the mean used in the Poisson distribution over transmission-attempt amounts: that is, on average, there are *G* transmission-attempts per frame-time.


will overlap with start of i's frame
will overlap with end of i's frame
node i frame
t_0-1
t_0
t_0+1

Overlapping frames in the pure ALOHA protocol. Frame-time is equal to 1 for all frames.

Pure Aloha efficiency

P(success by given node) = P(node transmits) $\cdot$

P(no other node transmits in $[t_0-1,t_0]$ $\cdot$

P(no other node transmits in $[t_0,t_0+1]$

$= p \cdot (1-p)^{\mathbf{N-1}} \cdot (1-p)^{\mathbf{N-1}}$

$= p \cdot (1-p)^{\mathbf{2(N-1)}}$

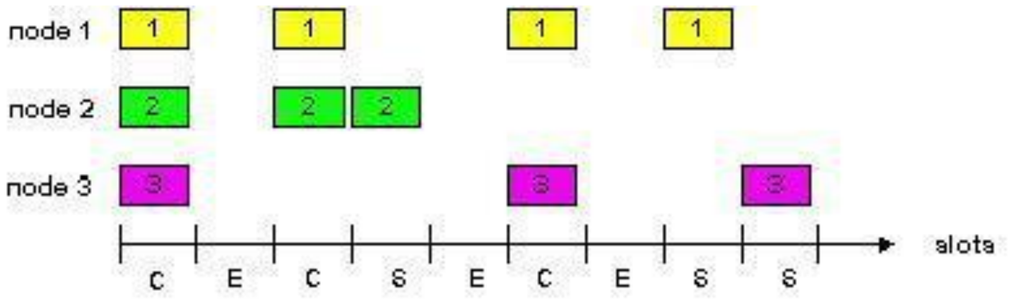
… choosing optimum p and then letting n -> ∞ ...

Efficiency = 1/(2e) = .18

SLOTTED ALOHA

- An improvement to the original ALOHA protocol was "Slotted ALOHA", which introduced discrete timeslots and increased the maximum throughput.

- A station can send only at the beginning of a timeslot, and thus collisions are reduced. In this case, we only need to worry about the transmission-attempts within 1 frame-time and not 2 consecutive frame-times, since collisions can only occur during each timeslot. Thus, the probability of there being zero transmission-attempts in a single timeslot is:



- Slotted ALOHA is used in low-data-rate tactical satellite communications networks by military forces, in subscriber-based satellite communications networks, mobile telephony call setup, and in the contactless RFID technologies.

Pros

- single active node can continuously transmit at full rate of channel
- highly decentralized: only slots in nodes need to be in sync
- simple

Cons

- collisions, wasting slots

- idle slots
- nodes may be able to detect collision in less than time to transmit packet
- clock synchronization

Efficiency is the long-run fraction of successful slots when there are many nodes, each with many frames to send

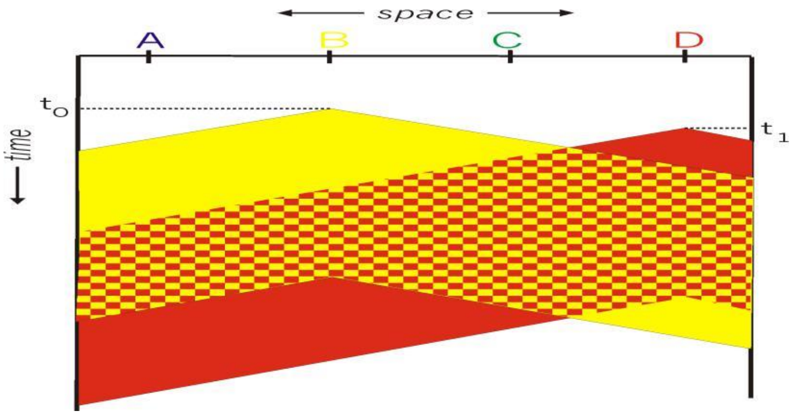
- Suppose N nodes with many frames to send, each transmits in slot with probability *p*
- prob that node 1 has success in a slot = $p(1-p)^{\mathbf{N-1}}$
- prob that any node has a success = $Np(1-p)^{\mathbf{N-1}}$
- For max efficiency with N nodes, find p* that maximizes $Np(1-p)^{\mathbf{N-1}}$
- For many nodes, take limit of $Np^*(1-p^*)^{\mathbf{N-1}}$ as N goes to infinity, gives 1/e = .37
- Efficiency is 37%, even with optimal p

CARRIER SENSE MULTIPLE ACCESS

Carrier Sense Multiple Access (**CSMA**) is a probabilistic Media Access Control (MAC) protocol in which a node verifies the absence of other traffic before transmitting on a shared transmission medium, such as an electrical bus, or a band of the electromagnetic spectrum.

"**Carrier Sense**" describes the fact that a transmitter uses feedback from a receiver that detects a carrier wave before trying to send. That is, it tries to detect the presence of an encoded signal from another station before attempting to transmit. If a carrier is sensed, the station waits for the transmission in progress to finish before initiating its own transmission.

"**Multiple Access**" describes the fact that multiple stations send and receive on the medium. Transmissions by one node are generally received by all other stations using the medium.



ADVANTAGES

- Fairly simple to implement
- Functional scheme that works

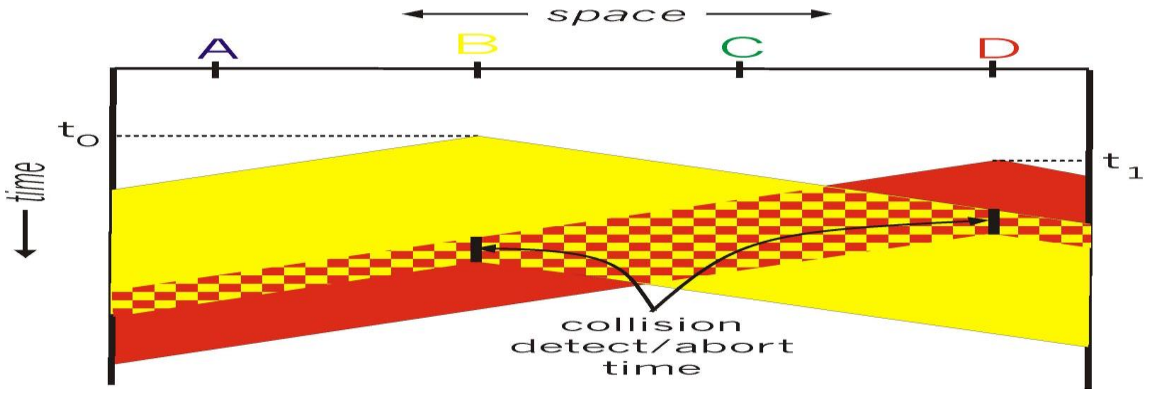
DISADVANTAGES

- Cannot recover from a collision (inefficient waste of medium time)

CSMA/CD (Collision Detection)

Carrier sense multiple access with collision detection (**CSMA/CD**) is a Media Access Control method in which.

- a carrier sensing scheme is used.
- a transmitting data station that detects another signal while transmitting a frame, stops transmitting that frame, transmits a **jam signal**, and then waits for a random time interval before trying to resend the frame.
- CSMA/CD is a modification of pure carrier sense multiple access (CSMA). CSMA/CD is used to improve CSMA performance by terminating transmission as soon as a collision is detected, thus shortening the time required before a retry can be attempted.



ALGORITHM

When a station wants to send some information, it uses the following algorithm.

Main procedure

1. Frame ready for transmission.
2. Is medium idle? If not, wait until it becomes ready
3. Start transmitting.
4. Did a collision occur? If so, go to collision detected procedure.
5. Reset retransmission counters and end frame transmission.

Collision detected procedure

1. Continue transmission until minimum packet time is reached to ensure that all receivers detect the collision.

2. Increment retransmission counter.
3. Was the maximum number of transmission attempts reached? If so, abort transmission.
4. Calculate and wait random backoff period based on number of collisions.
5. Re-enter main procedure at stage 1.

Methods for collision detection are media dependent, but on an electrical bus such as 10BASE-5 or 10BASE-2, collisions can be detected by comparing transmitted data with received data or by recognizing a higher than normal signal amplitude on the bus.

JAM SIGNAL

The **jam signal** is a signal that carries a 32-bit binary pattern sent by a data station to inform the other stations that they must not transmit.

ADVANTAGES

More efficient than basic CSMA

DISADVANTAGES

Requires ability to detect collisions

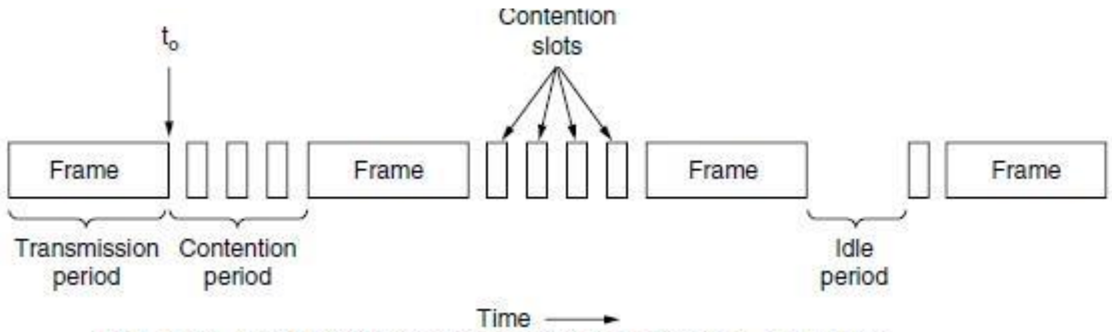


Figure 4-5. CSMA/CD can be in contention, transmission, or idle state.

BRIDGE

Bridges can be used to connect two or more LAN segments of the same type (e.g. Ethernet to Ethernet, or Token-Ring to Token-Ring). Like repeaters, bridges can extend the length of a network, but unlike repeaters they can also extend the capacity of a network, since each port on a bridge has its own MAC address. When bridges are powered on in an Ethernet network, they start to learn the network's topology by analysing the source addresses of incoming frames from all attached network segments (a process called backward learning).

Over a period of time, they build up a routing table . Unless the source and the destination are on different network segments, there is no need for the bridge to transfer an incoming frame to another network segment. If the source and the destination are on different segments, the bridge needs to be able to determine which segment the destination device belongs to.

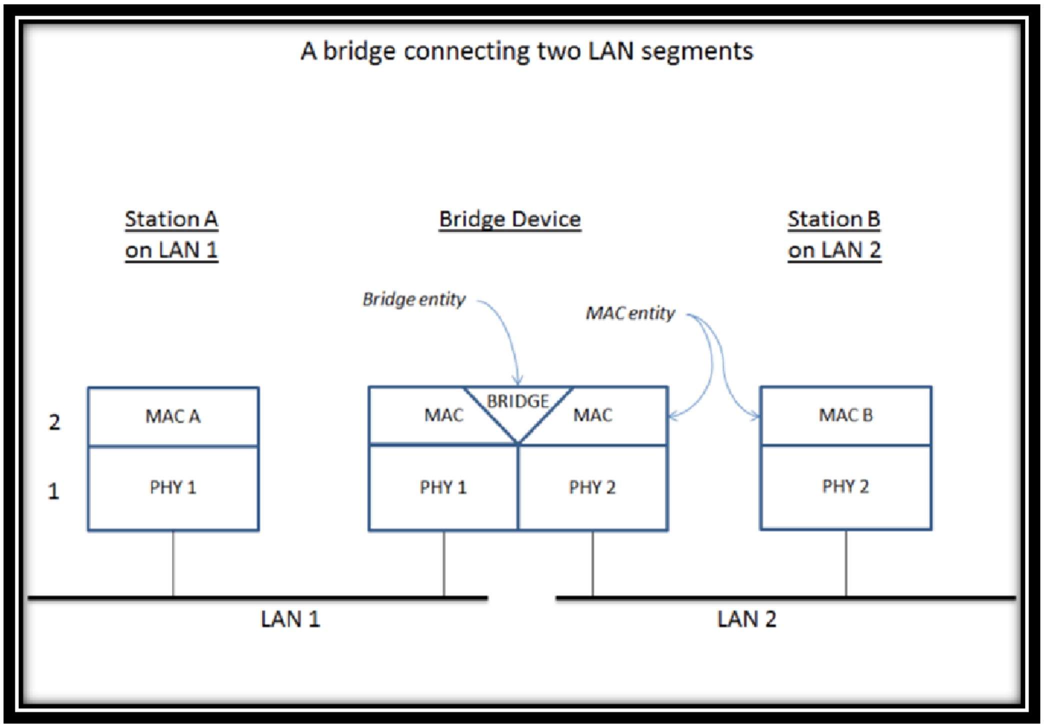


Fig: A high-level overview of network bridging, using the ISO/OSI layers and terminology

Types

There are four types of network bridging technologies: simple bridging, multiport bridging, learning or transparent bridging, and source route bridging.

Transparent bridging

A transparent bridge uses a forwarding database to send frames across network segments. The forwarding database is initially empty and entries in the database are built as the bridge receives frames. If an address entry is not found in the forwarding database, the frame is flooded to all other ports of the bridge, flooding the frame to all segments except the one from which it was received. By means of these flooded frames, the destination network will respond and a forwarding database entry will be created.

In the context of a two-port bridge, the forwarding database can be thought of as a filtering database. A bridge reads a frame's destination address and decides to either forward or filter. If the bridge determines that the destination node is on another segment on the network, it forwards (retransmits) the frame to that segment. If the destination address belongs to the same segment as the source address, the bridge filters (discards) the frame. As nodes transmit data through the bridge, the bridge establishes a filtering database of known MAC addresses and their locations on the network. The bridge uses its filtering database to determine whether a packet should be forwarded or filtered.

Simple bridging

The simple bridge connects two network segments. It typically operates transparently and decides on a packet-by-packet basis whether or not to forward from one network to the other. A store and forward technique is typically used such that, during forwarding, packet integrity is verified on the source network and CSMA/CD delays are accommodated on the destination network. In contrast to simple repeaters that were used to just extend the maximum reach of a segment, a bridge also reduces collisions by splitting the collision domain. Additionally, it lowers overall traffic by only forwarding those packets that are required to cross the bridge.

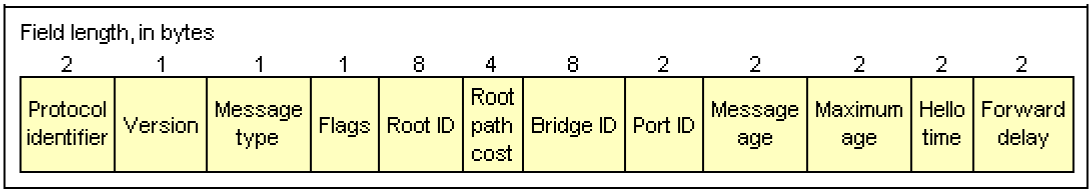
Multiport bridging

A multiport bridge connects multiple networks and operates transparently to decide on a packet-by-packet basis whether and where to forward. Like the simple bridge, a multiport bridge typically uses store and forward operation. The multiport bridge function is the basis for a network switch.

Source-Route Bridging (Token Ring)

The source-route bridging (SRB) algorithm was developed by IBM for Token Ring networks, and gets its name from the fact that routing information is placed in all inter-segment frames by the sending device. Bridges forward frames according to the routing information carried within the frame. A simple source-route bridging network is illustrated below.

Frame format



Field length, in bytes

2	1	1	1	8	4	8	2	2	2	2	2
Protocol identifier	Version	Message type	Flags	Root ID	Root path cost	Bridge ID	Port ID	Message age	Maximum age	Hello time	Forward delay

Fig: Bridge configuration message format

The fields of the bridge configuration message are described below.

Protocol identifier - contains the value zero.

Version - contains the value zero.

Message type - contains the value zero

Flags - only the first two bits are used. The topology-change bit , if set, signals a topology change, and the topology-change acknowledgment bit , if set, acknowledges receipt of a configuration message with the topology-change bit set.

Root ID - identifies the root bridge using its 2-byte priority followed by its 6-byte ID.

Root path cost - the root path cost from the bridge sending the configuration message to the root bridge.

Bridge ID - identifies the bridge sending the message using its 2-byte priority followed by its 6-byte ID.

Port ID - identifies the port from which the configuration message was sent.

Message age - indicates when the configuration message should be deleted.

Maximum age - contains the value zero.

Hello time - indicates the time period between root bridge configuration messages.

Forward delay - the time bridges should wait before transitioning to a new state after a topology change.

HUB

Hubs are used in Ethernet networks. A signal received at any port on the hub is retransmitted on all other ports. Network segments that employ hubs are often described as having a star topology, in which the hub forms the wiring centre of the star.

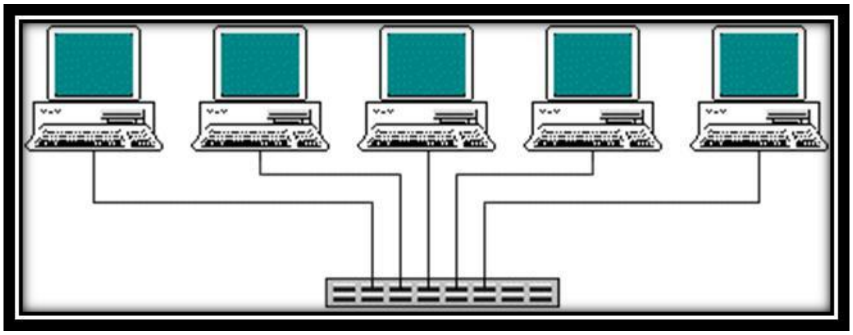


Fig: A hub in a star network configuration

Using a hub provides a degree of fault tolerance, because each network device has its own connection to the hub, and if a connection fails, only a single device is affected. Expanding the network is also easier, because many additional devices can be added to the network using a single hub, which is itself often connected to a network backbone. Hubs can be either active or passive. An active hub has its own power supply, and regenerates incoming frames before retransmitting them. Because signals are regenerated, each output port can connect a channel of up to 100 metres (the maximum allowed for twisted pair cables). Passive hubs simply relay the signal without regenerating it. Managed hubs allow administrators to enable or disable individual ports remotely, while intelligent hubs can autonomously close ports down if the occurrence of errors in transmitted packets exceeds a certain threshold.

Fig: A 24-port hub

What Hubs Do

Hubs and switches serve as a central connection for all of your network equipment and handles a data type known as frames. Frames carry your data. When a frame is received, it is amplified and then transmitted on to the port of the destination PC.

In a hub, a frame is passed along or "broadcast" to every one of its ports. It doesn't matter that the frame is only destined for one port. The hub has no way of distinguishing which port a frame should be sent to. Passing it along to every port ensures that it will reach its intended destination. This places a lot of traffic on the network and can lead to poor network response times.

SWITCH

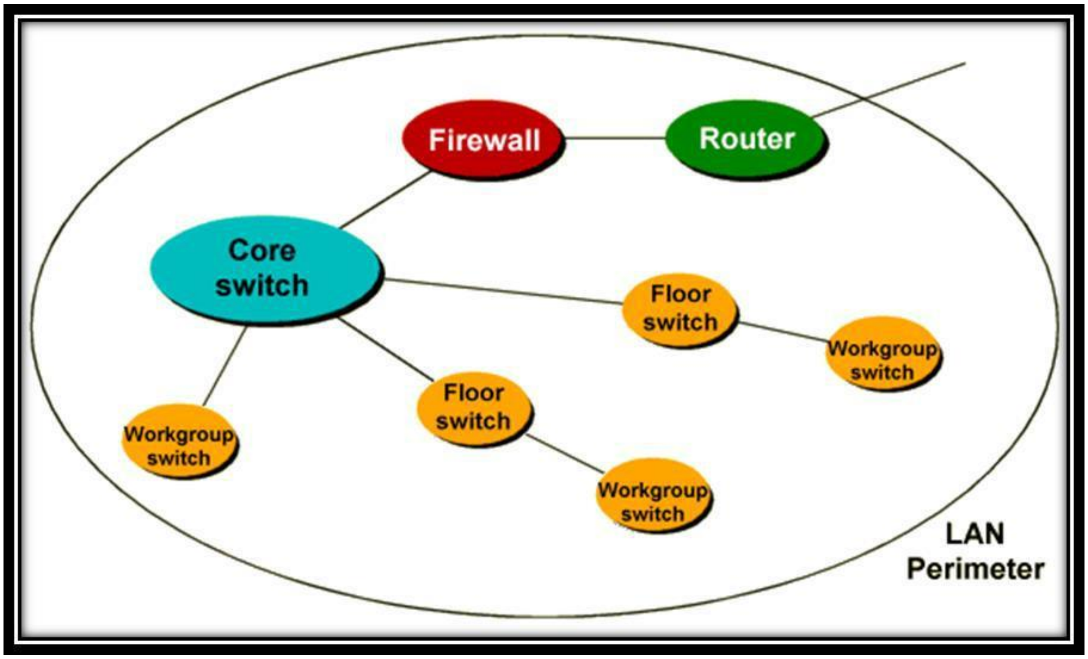


Fig: A core switch connects the high-level devices on the network

The switch is a relatively new network device that has replaced both hubs and bridges in LANs. A switch uses an internal address table to route incoming data frames via the port associated with their destination MAC address. Switches can be used to connect together a number of end-user devices such as workstations, or to interconnect multiple network segments.

A switch that interconnects end-user devices is often called a *workgroup switch*. Switches provide dedicated full-duplex links for every possible pairing of ports, effectively giving each attached device its own network segment This significantly reduces the number of intra-segment and inter-segment collisions.

The Hub and Switch Have Similar Roles

Each serves as a central connection for all of your network equipment and handles a data type known as frames. Frames carry your data. When a frame is received, it is amplified and then transmitted on to the port of the destination PC. The big difference between these two devices is in the method in which frames are being delivered.

In a hub, a frame is passed along or "broadcast" to every one of its ports. It doesn't matter that the frame is only destined for one port. The hub has no way of distinguishing which port a frame should be sent to. Passing it along to every port ensures that it will reach its intended destination. This places a lot of traffic on the network and can lead to poor network response times.

Additionally, a 10/100Mbps hub must share its bandwidth with each and every one of its ports. So when only one PC is broadcasting, it will have access to the maximum available bandwidth. If, however, multiple PCs are broadcasting, then that bandwidth will need to be divided among all of those systems, which will degrade performance.

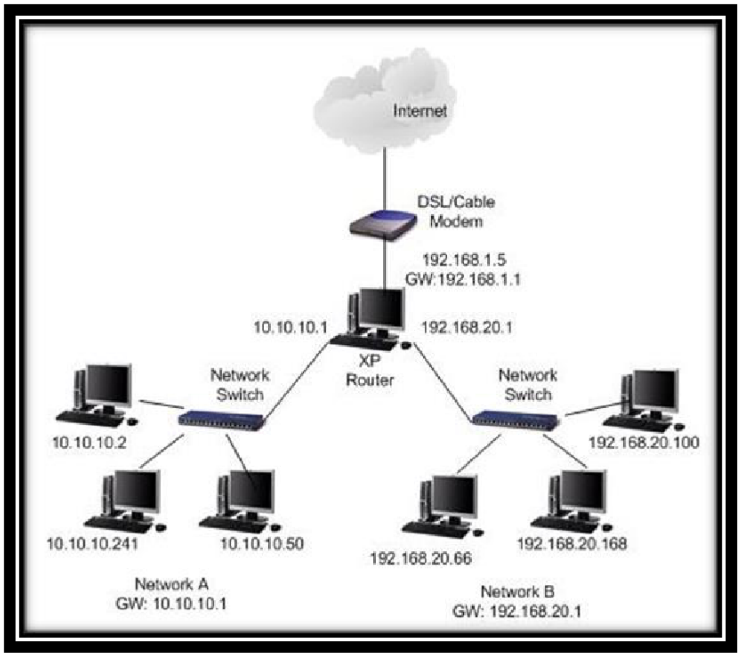
A switch, however, keeps a record of the MAC addresses of all the devices connected to it. With this information, a switch can identify which system is sitting on which port. So when a frame is received, it knows exactly which port to send it to, without significantly increasing network response times.

And, unlike a hub, a 10/100Mbps switch will allocate a full 10/100Mbps to each of its ports. So regardless of the number of PCs transmitting, users will always have access to the maximum amount of bandwidth. It's for these reasons a switch is considered to be a much better choice then a hub.

ROUTER

A network environment that consists of several interconnected networks employing different network protocols and architectures requires a sophisticated device to manage the flow of traffic between these diverse networks. Such a device, sometimes referred to as an intermediate system, but more commonly called a router, must be able to determine how to get incoming packets (or datagrams) to the destination network by the most efficient route.

Routers gather information about the networks to which they are connected, and can share this information with routers on other networks. The information gathered is stored in the router's internal routing table, and includes both the routing information itself and the current status of various network links. Routers exchange this routing information using special routing protocols.



Computers, and other end-user devices attached to networks that form part of an internetwork, are often called hosts or end-systems. A network host does not know how to forward a datagram to a host on another network, and so it will forward the datagram to its local router (or default gateway). A datagram may traverse a number of networks, and hence a number of routers, as it travels from an end-system on the source network to an end-system on the destination network. At each intermediate router, a decision is made as to what is the optimum next hop.

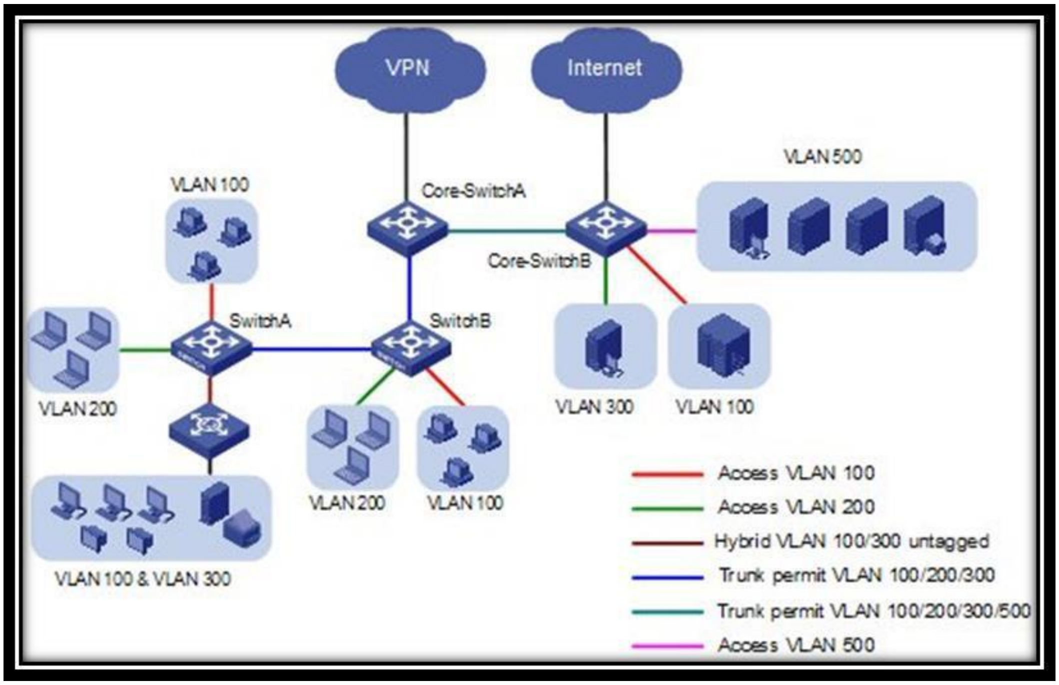
The process undertaken by the router in transferring the incoming datagram to one of its output ports in this way is called switching, and routers are at the heart of packet-switching networks. Unlike bridges and switches, routers do not concern themselves with MAC addresses, and instead examine the IP address contained within a datagram to determine the address of the destination network.

VIRTUAL LAN

In computer networking, a single layer-2 network may be partitioned to create multiple distinct broadcast domains, which are mutually isolated so that packets can only pass between them via one or more routers; such a domain is referred to as a virtual local area network, virtual LAN or VLAN.

This is usually achieved on switch or router devices. Simpler devices only support partitioning on a port level (if at all), so sharing VLANs across devices requires running dedicated cabling for each VLAN. More sophisticated devices can mark packets through tagging, so that a single interconnect (trunk) may be used to transport data for multiple VLANs.

Grouping hosts with a common set of requirements regardless of their physical location by VLAN can greatly simplify network design. A VLAN has the same attributes as a physical local area network (LAN), but it allows for end stations to be grouped together more easily even if they are not on the same network switch. VLAN membership can be configured through software instead of physically relocating devices or connections. Most enterprise-level networks today use the concept of virtual LANs. Without VLANs, a switch considers all interfaces on the switch to be in the same broadcast domain.



Protocols and design

IEEE 802.1Q

The protocol most commonly used today to configure VLANs is IEEE 802.1Q. The IEEE committee defined this method of multiplexing VLANs in an effort to provide multivendor VLAN support. Prior to the introduction of the 802.1Q standard, several proprietary protocols existed, such as Cisco's ISL (Inter-Switch Link) and 3Com's VLT (Virtual LAN Trunk). Cisco also implemented VLANs over FDDI by carrying VLAN information in an IEEE 802.10 frame header, contrary to the purpose of the IEEE 802.10 standard.

Both ISL and IEEE 802.1Q tagging perform "explicit tagging" - the frame itself is tagged with VLAN information. ISL uses an external tagging process that does not modify the existing Ethernet frame, while 802.1Q uses a frame-internal field for tagging, and therefore does modify the Ethernet frame. This internal tagging is what allows IEEE 802.1Q to work on both access and trunk links: frames are standard Ethernet, and so can be handled by commodity hardware.

Under IEEE 802.1Q, the maximum number of VLANs on a given Ethernet network is 4,094 (the 4,096 provided for by the 12-bit VID field minus reserved values 0x000 and 0xFFF). This does not impose the same limit on the number of IP subnets in such a network, since a single VLAN can contain multiple IP subnets. The VLAN limit is expanded to 16 million with Shortest Path Bridging.

Protocol-based VLAN’s

In a switch that supports protocol-based VLANs, traffic is handled on the basis of its protocol. Essentially, this segregates or forwards traffic from a port depending on the particular protocol of that traffic; traffic of any other protocol is not forwarded on the port.

For example, it is possible to connect the following to a given switch:

A host generating ARP traffic to port 10

A network with IPX traffic to port 20

A router forwarding IP traffic to port 30

If a protocol-based VLAN is created that supports IP and contains all three ports, this prevents IPX traffic from being forwarded to ports 10 and 30, and ARP traffic from being forwarded to ports 20 and 30, while still allowing IP traffic to be forwarded on all three ports.