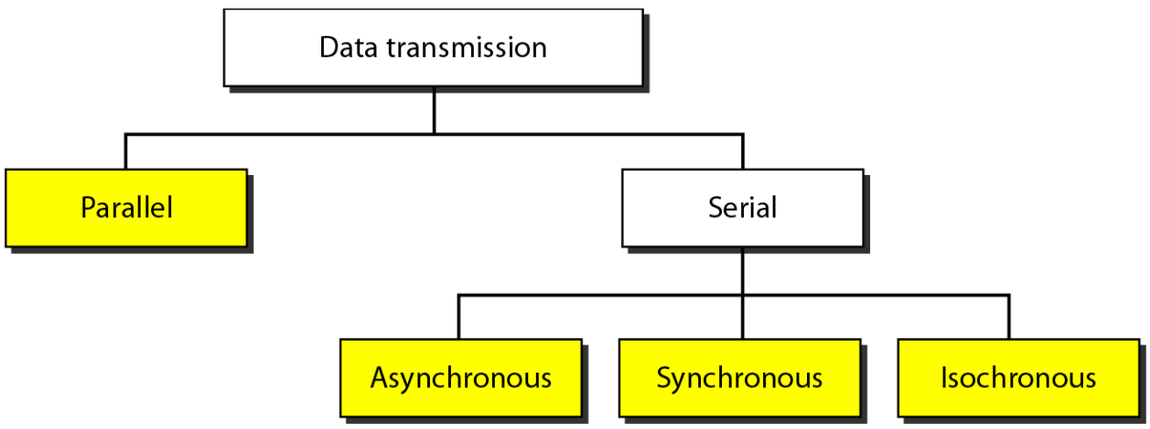
Unit IV

TRANSMISSION MODES

The transmission of binary data across a link can be accomplished in either parallel or serial mode. In parallel mode, multiple bits are sent with each clock tick. In serial mode, 1 bit is sent with each clock tick. While there is only one way to send parallel data, there are three subclasses of serial transmission: asynchronous, synchronous, and isochronous.



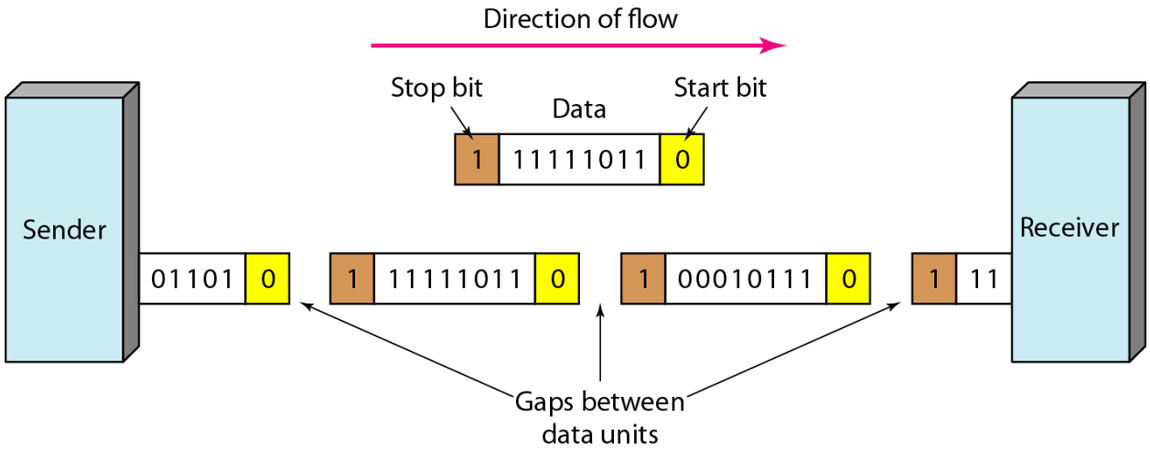
Asynchronous Transmission

Asynchronous transmission is so named because the timing of a signal is unimportant. Instead, information is received and translated by agreed upon patterns. As long as those patterns are followed, the receiving device can retrieve the information without regard to the rhythm in which it is sent. Patterns are based on grouping the bit stream into bytes. Each group, usually 8 bits, is sent along the link as a unit. The sending system handles each group independently, relaying it to the link whenever ready, without regard to a timer.

In asynchronous transmission, we send 1 start bit (0) at the beginning and 1 or more stop bits (1s) at the end of each byte. There may be a gap between each byte.

Asynchronous here means “asynchronous at the byte level, but the bits are still synchronized; their durations are the same.

Asynchronous transmission



The addition of stop and start bits and the insertion of gaps into the bit stream make asynchronous transmission slower than forms of transmission that can operate without the addition of control information. But it is cheap and effective, two advantages that make it an attractive choice for situations such as low-speed communication. For example, the connection of a keyboard to a computer is a natural application for asynchronous transmission. A user type's only one character at a time, types extremely slowly in data processing terms, and leaves unpredictable gaps of time between each character.

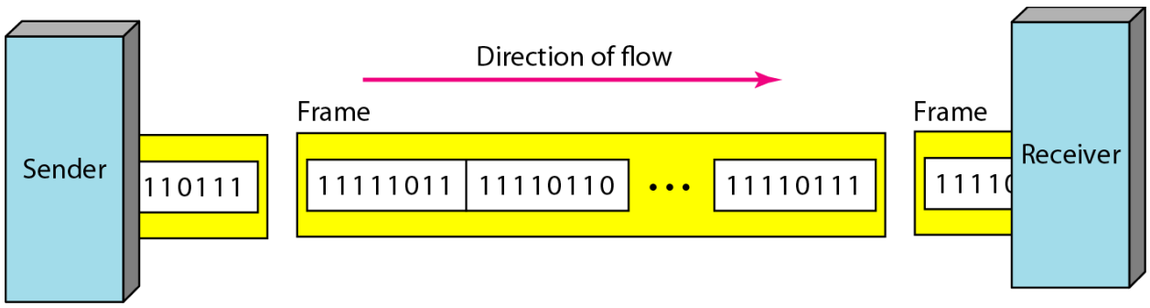
Synchronous Transmission

In synchronous transmission, we send bits one after another without start or stop bits or gaps. It is the responsibility of the receiver to group the bits.

Timing becomes very important, therefore, because the accuracy of the received information is completely dependent on the ability of the receiving device to keep an accurate count of the bits as they come in.

The advantage of synchronous transmission is speed. With no extra bits or gaps, synchronous transmission is faster than asynchronous transmission.

For this reason, it is more useful for high-speed applications such as the transmission of data from one computer to another. Byte synchronization is accomplished in the data link layer.



ERROR DETECTION

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

In error detection, we are looking only to see if any error has occurred. The answer is a simple Yes or No. We are not even interested in the number of errors. A single-bit error is the same for us as a burst error.

Detection Methods

- Parity Check
- Cyclical Redundancy Check (CRC)
- Checksum
- Parity and CRC are performed by data link layer; checksum performed by higher-layer protocols

Parity Check

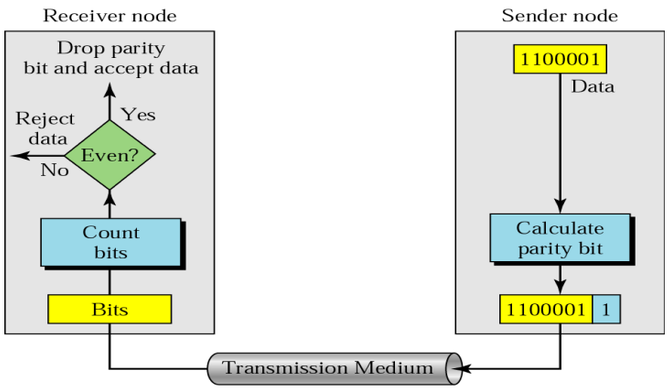
The most familiar error-detecting code is the simple parity check-code. In this code, a k-bit dataword is changed to an n-bit codeword where n= k+1. The extra bit, called the parity bit is selected to make the total number of 1s even or odd based on the scheme used parity check can be

- Even parity – In even-parity, a redundant bit (parity bit) is appended to every data unit so total number of 1 bits is even

  E.g., 1011 ® 101**11**

- Odd parity – In odd-parity, a redundant bit (parity bit) is appended to every data unit so total number of 1 bits is odd.

  E.g., 1011 ® 1011**0**



Suppose the sender wants to send the word *world*. In ASCII the five characters are coded as

1110111 1101111 1110010 1101100 1100100

The following shows the actual bits sent

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

Now suppose the word world in Example 1 is received by the receiver without being corrupted in transmission.

11101110 11011110 11100100 11011000 11001001

The receiver counts the 1s in each character and comes up with even numbers (6, 6, 4, 4, 4). The data are accepted.

Now suppose the word world in Example 1 is corrupted during transmission.

11111110 11011110 11101100 11011000 11001001

The receiver counts the 1s in each character and comes up with even and odd numbers (7, 6, 5, 4, 4). The receiver knows that the data are corrupted, discards them, and asks for retransmission.

Simple Parity Performance

- Can detect all single-bit errors
- May detect all burst errors as long as total number of bits changed is odd
- Cannot detect errors when total number of bits changed is even since parity check will pass even though errors had occurred

Cyclic Redundancy Check (CRC)

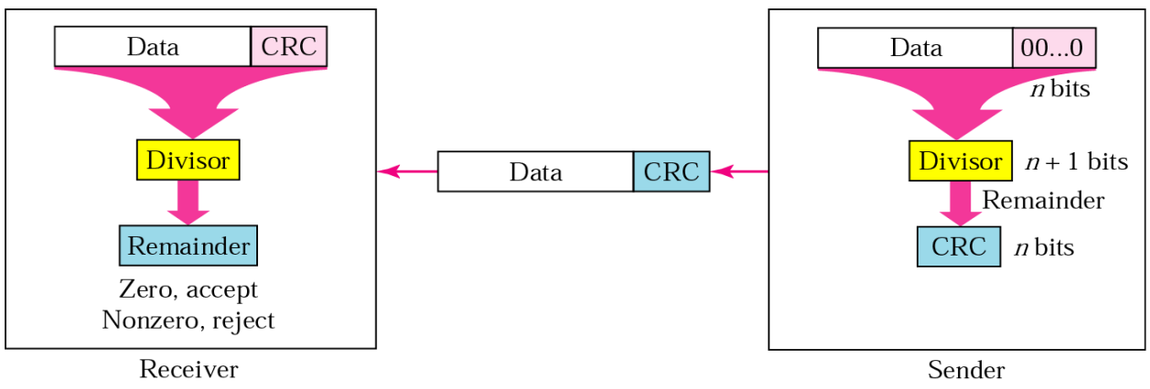
- Parity checks is based on addition and CRC is based on binary division

- A sequence of redundant bits (a CRC or CRC remainder) is appended to the end of the data unit
- These bits are later used in calculations to detect whether or not an error had occurred

CRC Steps

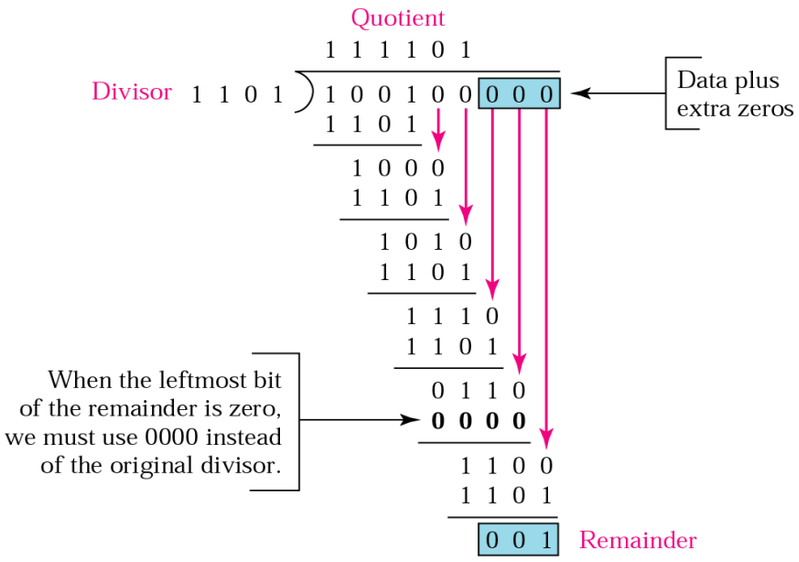
- On sender's end, data unit is divided by a predetermined divisor; remainder is the CRC
- When appended to the data unit, it should be exactly divisible by a second predetermined binary number
- At receiver's end, data stream is divided by same number
- If no remainder, data unit is assumed to be error-free
- Deriving the CRC
- A string of 0s is appended to the data unit; *n* is one less than number of bits in predetermined divisor
- New data unit is divided by the divisor using binary division; remainder is CRC
- CRC of *n* bits replaces appended 0s at end of data unit

CRC Generator and Checker



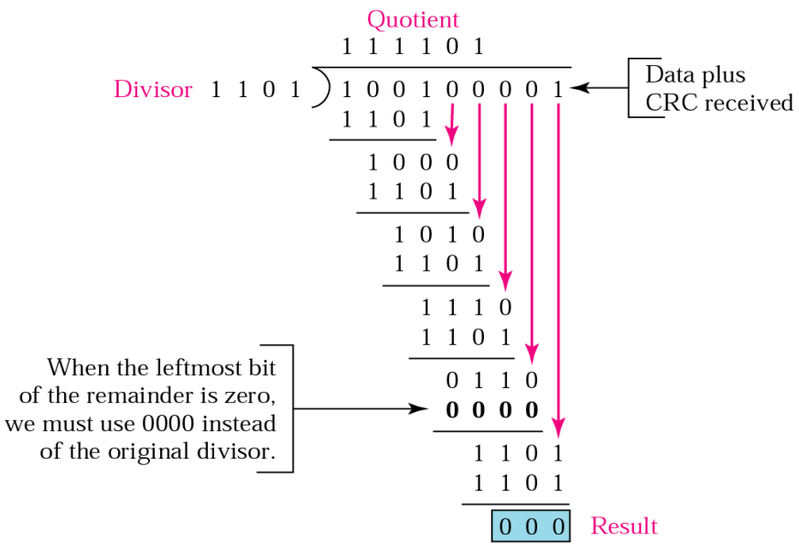
CRC Generator

- Uses modulo-2 division
- Resulting remainder is the CRC



CRC Checker

- Performed by receiver
- Data is appended with CRC
- Same modulo-2 division is applied
- If remainder is 0, data are accepted
- Otherwise, an error has occurred



Frame check sequence (FCS)

Structure of an Ethernet packet, including the FCS that terminates the Ethernet frame.

A frame check sequence (FCS) refers to the extra error-detecting code added to a frame in a communications protocol. Frames are used to send upper-layer data and ultimately the application data from a source to a destination.

The detection does not imply error recovery; for example, Ethernet specifies that a damaged frame should be discarded, but at the same time does not specify any action to cause the frame to be retransmitted. Other protocols, notably the Transmission Control Protocol (TCP), can notice the data loss and initiate error recovery.

All frames and the bits, bytes, and fields contained within them, are susceptible to errors from a variety of sources. The FCS field contains a number that is calculated by the source node based on the data in the frame. This number is added to the end of a frame that is sent. When the destination node receives the frame the FCS number is recalculated and compared with the FCS number included in the frame. If the two numbers are different, an error is assumed and the frame is discarded. The sending host computes a cyclic redundancy check on the entire frame and appends this as a trailer to the data. The receiving host recomputes the cyclic redundancy check on the frame using the same algorithm, and compares it to the received FCS. This way it can detect whether any data was lost or altered in transit. It may then discard the data, and request retransmission of the faulty frame.

The FCS is often transmitted in such a way that the receiver can compute a running sum over the entire frame, together with the trailing FCS, expecting to see a fixed result (such as zero) when it is correct. For Ethernet and other IEEE 802 protocols, this fixed result, also known as the *magic number* or *CRC32 residue*, is 0xC704DD7B. When transmitted and used in this way, FCS generally appears immediately before the frame-ending delimiter.

By far the most popular FCS algorithm is a cyclic redundancy check (CRC), used in Ethernet and other IEEE 802 protocols with 32 bits, in X.25 with 16 or 32 bits, in HDLC with 16 or 32 bits, in Frame Relay with 16 bits, in Point-to-Point Protocol (PPP) with 16 or 32 bits, and in other data link layer protocols.

Error correction

Error correction is the detection of errors and reconstruction of the original, error-free data.

Error correction may generally be realized in two different ways:

- *Automatic repeat request (ARQ)* (sometimes also referred to as *backward error correction*): This is an error control technique whereby an error detection scheme is combined with requests for retransmission of erroneous data. Every block of data received is checked using the error detection code used, and if the check fails, retransmission of the data is requested – this may be done repeatedly, until the data can be verified.

- *Forward error correction (FEC)*: The sender encodes the data using an *error-correcting code (ECC)*prior to transmission. The additional information (redundancy) added by the code is used by the receiver to recover the original data. In general, the reconstructed data is what is deemed the "most likely" original data.

ARQ and FEC may be combined, such that minor errors are corrected without retransmission, and major errors are corrected via a request for retransmission: this is called *hybrid automatic repeat-request (HARQ)*.

Automatic repeat request (ARQ)

Automatic Repeat request (ARQ) is an error control method for data transmission that makes use of error-detection codes, acknowledgment and/or negative acknowledgment messages, and timeouts to achieve reliable data transmission.
An *acknowledgment* is a message sent by the receiver to indicate that it has correctly received a data frame.
Usually, when the transmitter does not receive the acknowledgment before the timeout occurs (i.e., within a reasonable amount of time after sending the data frame), it retransmits the frame until it is either correctly received or the error persists beyond a predetermined number of retransmissions.
Three types of ARQ protocols are
Stop-and-wait ARQ,
Go-Back-N ARQ, and
Selective Repeat ARQ.

Stop & Wait Protocol

In this method of flow control, the sender sends a single frame to receiver & waits for an acknowledgment.

• The next frame is sent by sender only when acknowledgment of previous frame is received.

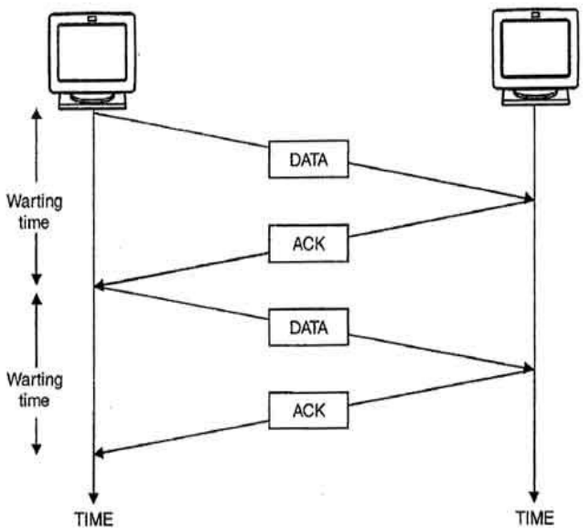
• This process of sending a frame & waiting for an acknowledgment continues as long as the sender has data to send.

• To end up the transmission sender transmits end of transmission (EOT) frame.

• The main advantage of stop & wait protocols is its accuracy. Next frame is transmitted only when the first frame is acknowledged. So there is no chance of frame being lost.

• The main disadvantage of this method is that it is inefficient. It makes the transmission process slow. In this method single frame travels from source to destination and single acknowledgment travels from destination to source. As a result each frame sent and received uses the entire time needed to traverse the link. Moreover, if two devices are distance apart, a lot of time is

wasted waiting for ACKs that leads to increase in total transmission time.



Stop & Wait Method.

Go- Back- N ARQ

Go-Back-N ARQ is a specific instance of the automatic repeat request (ARQ) protocol, in which the sending process continues to send a number of frames specified by a *window size* even without receiving an acknowledgement (ACK) packet from the receiver. It is a special case of the general sliding window protocol with the transmit window size of N and receive window size of 1. It can transmit N frames to the peer before requiring an ACK.

The receiver process keeps track of the sequence number of the next frame it expects to receive, and sends that number with every ACK it sends. The receiver will discard any frame that does not have the exact sequence number it expects (either a duplicate frame it already acknowledged or an out-of-order frame it expects to receive later) and will resend an ACK for the last correct in-order frame. Once the sender has sent all of the frames in its *window*, it will detect that all of the frames since the first lost frame are *outstanding*, and will go back to the sequence number of the last ACK it received from the receiver process and fill its window starting with that frame and continue the process over again.

Go-Back-N ARQ is a more efficient use of a connection than Stop-and-wait ARQ, since unlike waiting for an acknowledgement for each packet, the connection is still being utilized as packets are being sent. In other words, during the time that would otherwise be spent waiting, more packets are being sent. However, this method also results in sending frames multiple times – if any frame was lost or damaged, or the ACK acknowledging them was lost or damaged, then that frame and all following frames in the window (even if they were received without error) will be re-sent. To avoid this, Selective Repeat ARQ can be used.

Pseudocode

These examples assume an infinite number of sequence and request numbers.

```
N  = window size
Rn = request number
Sn = sequence number
Sb = sequence base
Sm = sequence max

Receiver:
Rn = 0
Do the following forever:
If the packet received = Rn and the packet is error free
        Accept the packet and send it to a higher layer
        Rn = Rn + 1
        Send a Request for Rn
Else
        Refuse packet
        Send a Request for Rn

Sender:
Sb = 0
Sm = N − 1
Repeat the following steps forever:
1. If you receive a request number where Rn > Sb
        Sm = Sm + (Rn − Sb)
        Sb = Rn
2.  If no packet is in transmission,
        Transmit a packet where Sb <= Sn <= Sm.
        Packets are transmitted in order.
```

Choosing a Window Size (N)

There are a few things to keep in mind when choosing a value for N:

1. The sender must not transmit too fast. N should be bounded by the receiver's ability to process packets.
2. N must be smaller than the number of sequence numbers (if they are numbered from zero to N) to verify transmission in cases of any packet (any data or ACK packet) being dropped.
3. Given the bounds presented in (1) and (2), choose N to be the largest number possible.

Selective Repeat ARQ

Selective Repeat ARQ / Selective Reject ARQ is a specific instance of the Automatic Repeat-Request (ARQ) protocol used to solve sequence number dilemma in communications.

Selective Repeat is part of the automatic repeat-request (ARQ). With selective repeat, the sender sends a number of frames specified by a window size even without the need to wait for individual ACK from the receiver as in Go-Back-N ARQ. The

receiver may selectively reject a single frame, which may be retransmitted alone; this contrast with other forms of ARQ, which must send every frame from that point again. The receiver accepts out-of-order frames and buffers them. The sender individually retransmits frames that have timed out.

It may be used as a protocol for the delivery and acknowledgement of message units, or it may be used as a protocol for the delivery of subdivided message sub-units.

When used as the protocol for the delivery of **messages**, the sending process continues to send a number of frames specified by a *window size* even after a frame loss. Unlike Go-Back-N ARQ, the receiving process will continue to accept and acknowledge frames sent after an initial error; this is the general case of the sliding window protocol with both transmit and receive window sizes greater than 1.

The receiver process keeps track of the sequence number of the earliest frame it has not received, and sends that number with every acknowledgement (ACK) it sends. If a frame from the sender does not reach the receiver, the sender continues to send subsequent frames until it has emptied its *window*. The receiver continues to fill its receiving window with the subsequent frames, replying each time with an ACK containing the sequence number of the earliest missing frame. Once the sender has sent all the frames in its *window*, it re-sends the frame number given by the ACKs, and then continues where it left off.

The size of the sending and receiving windows must be equal, and half the maximum sequence number (assuming that sequence numbers are numbered from 0 to $n-1$) to avoid miscommunication in all cases of packets being dropped. To understand this, consider the case when all ACKs are destroyed. If the receiving window is larger than half the maximum sequence number, some, possibly even all, of the packets that are resent after timeouts are duplicates that are not recognized as such. The sender moves its window for every packet that is acknowledged.

When used as the protocol for the delivery of subdivided messages it works somewhat differently. In non-continuous channels where messages may be variable in length, standard ARQ or Hybrid ARQ protocols may treat the message as a single unit. Alternately selective retransmission may be employed in conjunction with the basic ARQ mechanism where the message is first subdivided into sub-blocks (typically of fixed length) in a process called packet segmentation. The original variable length message is thus represented as a concatenation of a variable number of sub-blocks. While in standard ARQ the message as a whole is either acknowledged (ACKed) or negatively acknowledged (NAKed), in ARQ with selective transmission the ACK response would additionally carry a bit flag indicating the identity of each sub-block successfully received. In ARQ with selective retransmission of sub-divided messages each retransmission diminishes in length, needing to only contain the sub-blocks that were linked.

In most channel models with variable length messages, the probability of error-free reception diminishes in inverse proportion with increasing message length. In other words it's easier to receive a short message than a longer message. Therefore standard ARQ techniques involving variable length messages have increased difficulty delivering longer messages, as each repeat is the full length. Selective re-transmission applied to variable length messages completely eliminates the difficulty in delivering longer messages, as successfully delivered sub-blocks are retained after each transmission, and the number of outstanding sub-blocks in following transmissions diminishes. Selective Repeat is implemented in UDP transmission.

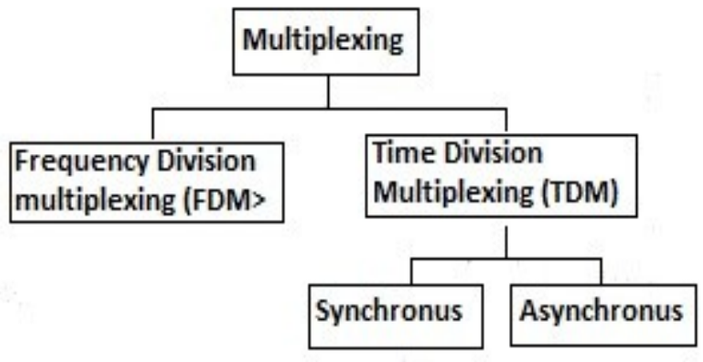
Multiplexing

Multiplexing is the set of techniques that allows the simultaneous transmission of multiple signals across a single data link. Whenever the transmission capacity of a medium linking two devices is greater than the transmission needs of the devices, the link can be shared in order to maximize the utilization of the link, such as one cable can carry a hundred channels of TV.

Type of multiplexing

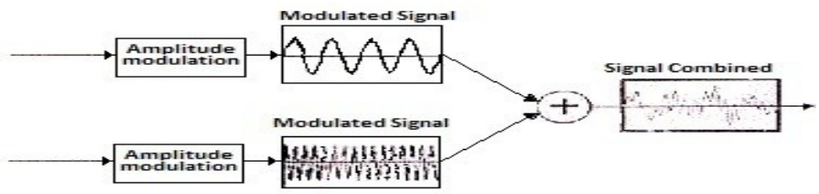
There are two basic techniques:

1. Frequency-Division Multiplexing (FDM)
2. Time-Division Multiplexing (TDM)
    1. Synchronous TDM
    2. A-Synchronous TDM

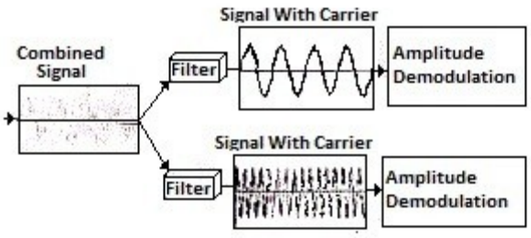


FREQUENCY-DIVISION MULTIPLEXING (FDM)

In FDM, signals generated by each sending device modulate different carrier frequencies. These modulated signals are then combined into a single composite signal that can be transported by the link. The carrier frequencies have to be different enough to accommodate the modulation and demodulation signals. The figure below illustrates the FDM multiplexing process. The multiplexing process starts by applying amplitude modulation into each signal by using different carrier frequencies . Then both signals are combined



In demultiplexing process, we use filters to decompose the multiplexed signal into its constituent component signals. Then each signal is passed to an amplitude demodulation process to separate the carrier signal from the message signal. Then, the message signal is sent to the waiting receiver. The process of demultiplexing is shown in the figure.

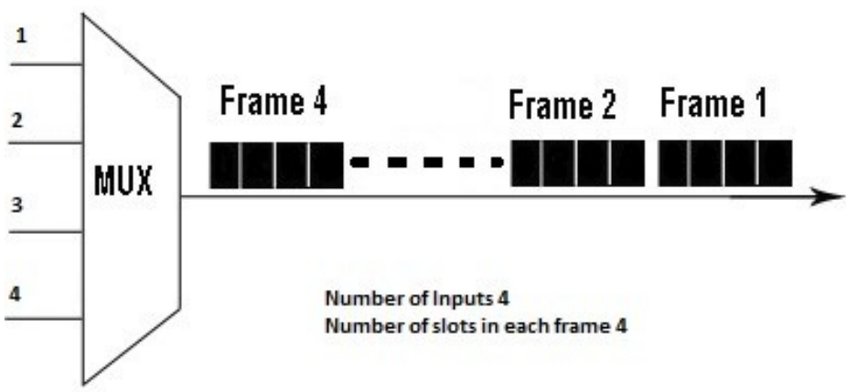


TIME-DIVISION MULTIPLEXING (TDM)

In the time-division multiplexing, multiple transmissions can occupy a single link by subdividing them and interleaving the portions. We say that TDM is a round robin use of a frequency. TDM can be implemented in two ways: synchronous TDM and asynchronous TDM.

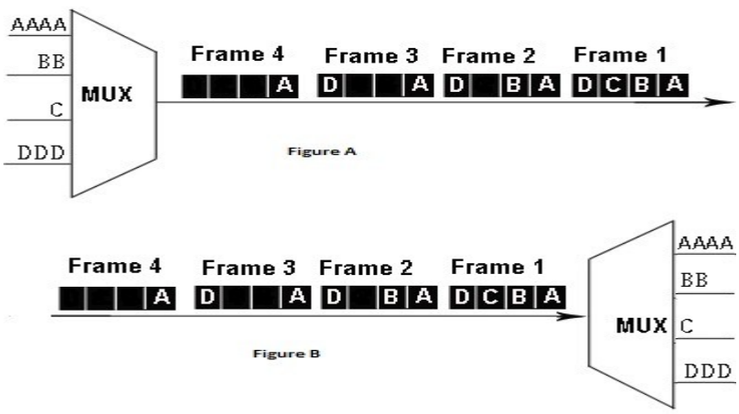
1. **Synchronous TDM**

The multiplexer allocates exactly the same time slot to each device at all times, whether or not a device has anything to transmit. Time slot 1, for example, is assigned to device 1 alone and cannot be used by any other device as shown in the figure



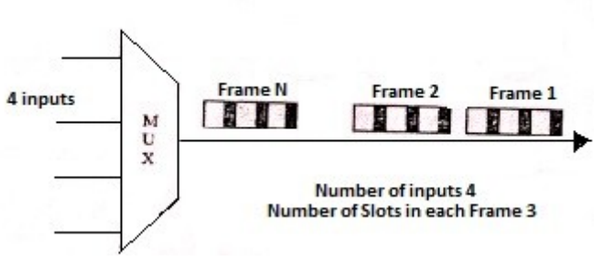
In synchronous TDM, a frame consists of one complete cycle of time slots. Thus the number of slots in frame is equal to the number of inputs.

Figures (A & B) below are an example of how/the synchronous TDM works.



2. Asynchronous TDM

In asynchronous TDM, each slot in a frame is not dedicated to the fix device. Each slot contains an index of the device to be sent to and a message. Thus, the number of slots in a frame is not necessary to be equal to the number of input devices. More than one slot in a frame can be allocated for an input device Asynchronous TDM allows maximization the link. It allows a number of lower speed input lines to be multiplexed to a single higher speed line. As shown in the figure



In asynchronous TDM, a frame contains a fix number of time slots. Each slot has an index of which device to receive.

Figures (A& B) are examples of how asynchronous TDM works.