UNIT -II
Data Link Layer

2.1 Framing

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 datagrams 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.

Since the physical layer merely accepts and transmits a stream of bits without any regard to meaning or structure, it is upto the data link layer to create and recognize frame boundaries. This can be accomplished by attaching special bit patterns to the beginning and end of the frame. If these bit patterns can accidentally occur in data, special care must be taken to make sure these patterns are not incorrectly interpreted as frame delimiters. The four framing methods that are widely used are

- Character count
- Starting and ending characters, with character stuffing
- Starting and ending flags, with bit stuffing
- Physical layer coding violations

Character Count

This 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. The disadvantage is that if the count is garbled by a transmission error, the destination will lose synchronization and will be unable to locate the start of the next frame. So, this method is rarely used.

Character stuffing

In the second method, each frame starts with the ASCII character sequence DLE STX and ends with the sequence DLE ETX.(where DLE is Data Link Escape, STX is Start of TeXt and ETX is End of TeXt.) This method overcomes the drawbacks of the character count method. If the destination ever loses synchronization, it only has to look for DLE STX and DLE ETX characters. If however, binary data is

being transmitted then there exists a possibility of the characters DLE STX and DLE ETX occurring in the data. Since this can interfere with the framing, a technique called character stuffing is used. The sender's data link layer inserts an ASCII DLE character just before the DLE character in the data. The receiver's data link layer removes this DLE before this data is given to the network layer. However character stuffing is closely associated with 8-bit characters and this is a major hurdle in transmitting arbitrary sized characters.

Bit stuffing

The third method allows data frames to contain an arbitrary number of bits and allows character codes with an arbitrary number of bits per character. At the start and end of each frame is a flag byte consisting of the special bit pattern 01111110. Whenever the sender's data link layer encounters five consecutive 1s in the data, it automatically stuffs a zero bit into the outgoing bit stream. This technique is called bit stuffing. When the receiver sees five consecutive 1s in the incoming data stream, followed by a zero bit, it automatically destuffs the 0 bit. The boundary between two frames can be determined by locating the flag pattern.

Physical layer coding violations

The final framing method is physical layer coding violations and is applicable to networks in which the encoding on the physical medium contains some redundancy. In such cases normally, a 1 bit is a high-low pair and a 0 bit is a low-high pair. The combinations of low-low and high-high which are not used for data may be used for marking frame boundaries.

2.2 HDLC(High Level Data Link Control) Protocol

The HDLC protocol is a general purpose protocol which operates at the data link layer of the OSI reference model. The protocol uses the services of a physical layer, and provides either a best effort or reliable communications path between the transmitter and receiver (i.e. with acknowledged data transfer). The type of service provided depends upon the HDLC mode which is used.

Each piece of data is encapsulated in an HDLC frame by adding a trailer and a header. The header contains an HDLC address and an HDLC control field. The trailer is found at the end of the frame, and contains a Cyclic Redundancy Check (CRC) which detects any errors which may occur during transmission. The frames are separated by HDLC flag sequences which are transmitted between each frame and whenever there is no data to be transmitted.

It is a transmission protocol used at the data link layer (layer 2) of the OSI seven layer model for data communications. The HDLC protocol embeds information in a data frame that allows devices to control data flow and correct errors. HDLC is an ISO standard developed from the Synchronous Data Link Control (SDLC) standard proposed by IBM in the 1970's.

For any HDLC communications session, one station is designated primary and the other secondary. A session can use one of the following connection modes, which determine how the primary and secondary stations interact.

Normal unbalanced: The secondary station responds only to the primary station.
Asynchronous: The secondary station can initiate a message.
Asynchronous balanced: Both stations send and receive over its part of a duplex line. This mode is used for X.25 packet-switching networks.

HDLC Frame Structure:

The HDLC frame consists of Flag, Address, Control, Data, and CRC fields as shown. The bit length of each field is given below:

Flag (both opening and closing flags): 8 bits (01111110 or 7E hex)

Address: It is normally 8 or 16 bits in length. A leading 'zero' bit (MSB) indicates a unicast message; the remaining bits provide the destination node address. A leading 'one' bit (MSB) location indicates multicast message, the remaining bits provide the group address.

Control: The field is 8 bits, or 16 bits wide and indicates whether the frame is a Control or Data frame. The field contains sequence number (hdlc frames are numbered to ensure delivery), poll (you need to reply) and final (indicating that this is the last frame) bits.

Data (Payload): This is the information that is carried from node to node. This is a variable field. Sometimes padded with extra bits to provide fixed length.

FCS (Frame Check Sequence) or CRC (Cyclic Redundancy Code): It is normally 16 bits wide. Frame Check Sequence is used to verify the data integrity. If the FCS fails, the frame is discarded.

Closing Flag: It is same as Opening Flag.

If no prior care is taken, it is possible that flag character (01111110) is present in data field. If present, then it will wrongly be interpreted as end of frame. To avoid this ambiguity, a transmitter will force a '0' bit after encountering 5 continuous 1s. At the receiving end, the receiver drops the '0' bit when encountered with 5 continuous 1s, and continues with the next bit. This way, the flag pattern (01111110) is avoided in the data field.

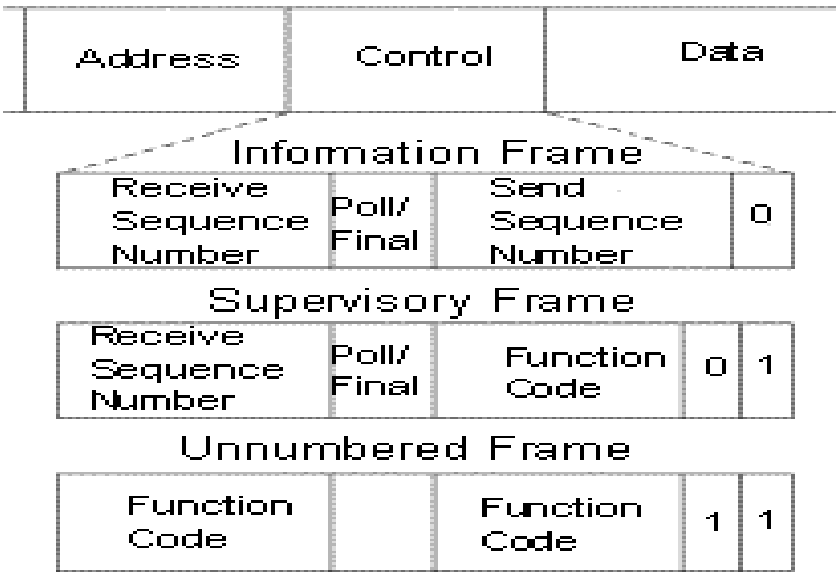
Normally, synchronous links transmit all the time. But, useful information may not be present at all times. Idle flags [11111111] may be sent to fill the gap between useful frames. Alternatively, a series of flags [01111110] may be transmitted to fill gaps between frames instead of transmitting idle flags [11111111]. Continuous transmission of signals is required to keep both the transmitting and receiving nodes synchronized.

Ex.: **frame**...*flag*...*flag*...*flag*...**frame**..*flag*..*flag*..**frame**...**frame**...

HDLC Frame Types

The control field in HDLC is also used to indicate the frame type. There are three types of frames supported by HDLC. These are:

I Frames: These are information frames, and contain user data
S Frames: These are supervisory frames, and contain contain commands and responses
U Frames: These are un-numbered frames, and typically contain commands and responses.



I Frames are sequentially numbered, carry user data, poll and final bits, and message acknowledgements.

S Frames performs any retransmission requests, and other supervisory controls.

U Frames can be used to initialize secondaries.

2.3 Error Detection and Correction

There are many reasons such as noise, cross-talk etc., which may help data to get corrupted during transmission. The upper layers work on some generalized view of network architecture and are not aware of actual hardware data processing. Hence, the upper layers expect error-free transmission between the systems. Most of the applications would not function expectedly if they receive erroneous data. Applications such as voice and video may not be that affected and with some errors they may still function well.

Data-link layer uses some error control mechanism to ensure that frames (data bit streams) are transmitted with certain level of accuracy. But to understand how errors is controlled, it is essential to know what types of errors may occur.

Types of Errors : There may be three types of errors

- **Single bit error**

  In a frame, there is only one bit, anywhere though, which is corrupt.

- **Multiple bits error**

  Frame is received with more than one bits in corrupted state.

- **Burst error**

  Frame contains more than1 consecutive bits corrupted.

Error control mechanism may involve two possible ways:

- Error detection
- Error correction

Error Detection

Errors in the received frames are detected by means of Parity Check and Cyclic Redundancy Check (CRC). In both cases, few extra bits are sent along with actual data to confirm that bits received at other end are same as they were sent. If the counter-check at receiver' end fails, the bits are considered corrupted.

(a) Parity Check

One extra bit is sent along with the original bits to make number of 1s either even in case of even parity, or odd in case of odd parity.

The sender while creating a frame counts the number of 1s in it. For example, if even parity is used and number of 1s is even then one bit with value 0 is added. This way number of 1s remains even. If the number of 1s is odd, to make it even a bit with value 1 is added.

The receiver simply counts the number of 1s in a frame. If the count of 1s is even and even parity is used, the frame is considered to be not-corrupted and is accepted. If the count of 1s is odd and odd parity is used, the frame is still not corrupted.

If a single bit flips in transit, the receiver can detect it by counting the number of 1s. But when more than one bits are erroneous, then it is very hard for the receiver to detect the error.

(b)Cyclic Redundancy Check (CRC)

CRC is a different approach to detect if the received frame contains valid data. This technique involves binary division of the data bits being sent. The divisor is generated using polynomials. The sender performs a division operation on the bits being sent and calculates the remainder. Before sending the actual bits, the sender adds the remainder at the end of the actual bits. Actual data bits plus the remainder is called a codeword. The sender transmits data bits as codewords. At the other end, the receiver performs division operation on codewords using the same CRC divisor. If the remainder contains all zeros the data bits are accepted, otherwise it is considered as there some data corruption occurred in transit.

Stronger kind of error-detecting code is in widespread use at the link layer: the CRC (Cyclic Redundancy Check), also known as a polynomial code.

Polynomial codes are based upon treating bit strings as representations of polynomials with coefficients of 0 and 1 only. A k-bit frame is regarded as the coefficient list for a polynomial with k terms, ranging from xk−1 to x0xk−1 to x0.

For example, 110001 has 6 bits and thus represents a six-term polynomial with coefficients 1, 1, 0, 0, 0, and 1: 1x5+1x4+0x3+0x2+0x1+1x01x5+1x4+0x3+0x2+0x1+1x0.

When the polynomial code method is employed, the sender and receiver must agree upon a generator polynomial, G(x), in advance. Both the high- and low order bits of the generator must be 1.

To compute the CRC for some frame with m bits corresponding to the polynomial M(x), the frame must be longer than the generator polynomial. The idea is to append a CRC to the end of the frame in such a way that the polynomial represented by the checksummed frame is divisible by G(x).

When the receiver gets the checksummed frame, it tries dividing it by G(x). If there is a remainder, there has been a transmission error.

The algorithm for computing the CRC is as follows:

-Let r be the degree of G(x). Append r zero bits to the low-order end of the frame so it now contains m + r bits and corresponds to the polynomial xr M(x).

-Divide the bit string corresponding to G(x) into the bit string corresponding to xr M(x), using modulo 2 divisions.

-Subtract the remainder (which is always r or fewer bits) from the bit string corresponding to xr M(x) using modulo 2 subtraction. The result is the check summed frame to be transmitted. Call its polynomial T(x).Figure below illustrates the calculation for a frame 1101011111 using the generator : G(x) = x4 + x + 1.

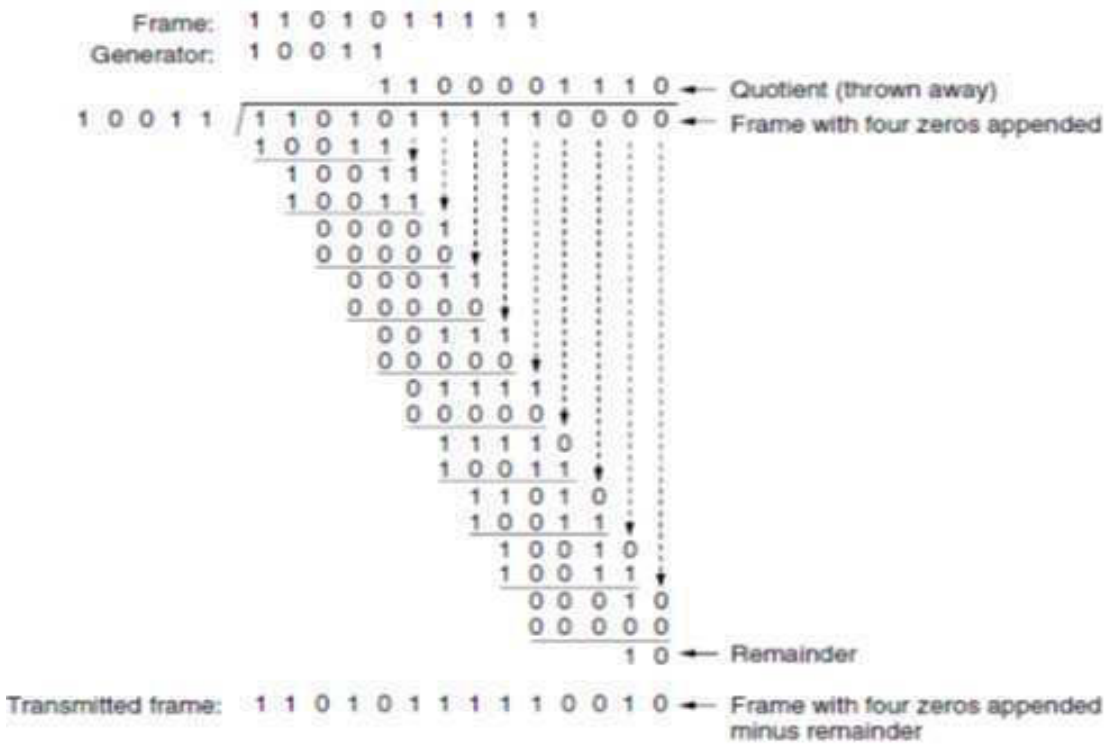


Fig. Example calculation of CRC

Error Correction

In the digital world, error correction can be done in two ways:

- **Backward Error Correction** When the receiver detects an error in the data received, it requests back the sender to retransmit the data unit.
- **Forward Error Correction** When the receiver detects some error in the data received, it executes error-correcting code, which helps it to auto-recover and to correct some kinds of errors.

The first one, Backward Error Correction, is simple and can only be efficiently used where retransmitting is not expensive. For example, fiber optics. But in case of wireless transmission retransmitting may cost too much. In the latter case, Forward Error Correction is used.

2.4 ARQ (Automatic Repeat Request)

Data-link layer is responsible for implementation of point-to-point flow and error control mechanism.

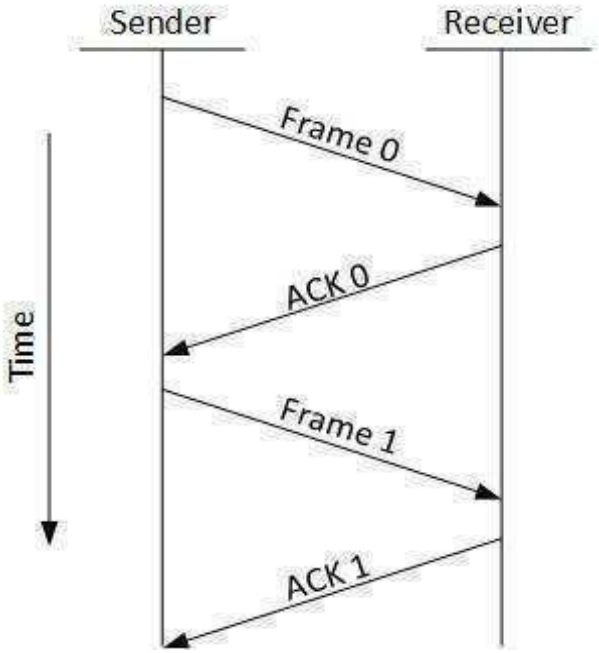
Flow Control

When a data frame (Layer-2 data) is sent from one host to another over a single medium, it is required that the sender and receiver should work at the same speed. That is, sender sends at a speed on which the receiver can process and accept the data. What if the speed (hardware/software) of the sender or receiver differs? If sender is sending too fast the receiver may be overloaded, (swamped) and data may be lost.

Two types of mechanisms can be deployed to control the flow:

Stop and Wait

This flow control mechanism forces the sender after transmitting a data frame to stop and wait until the acknowledgement of the data-frame sent is received.



Sliding Window

In this flow control mechanism, both sender and receiver agree on the number of data-frames after which the acknowledgement should be sent. As we learnt, stop and wait flow control mechanism wastes resources, this protocol tries to make use of underlying resources as much as possible.

Error Control

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

Error detection: The sender and receiver, either both or any, must ascertain that there is some error in the transit.

Positive ACK: When the receiver receives a correct frame, it should acknowledge it.

Negative ACK: When the receiver receives a damaged frame or a duplicate frame, it sends a NACK back to the sender and the sender must retransmit the correct frame.

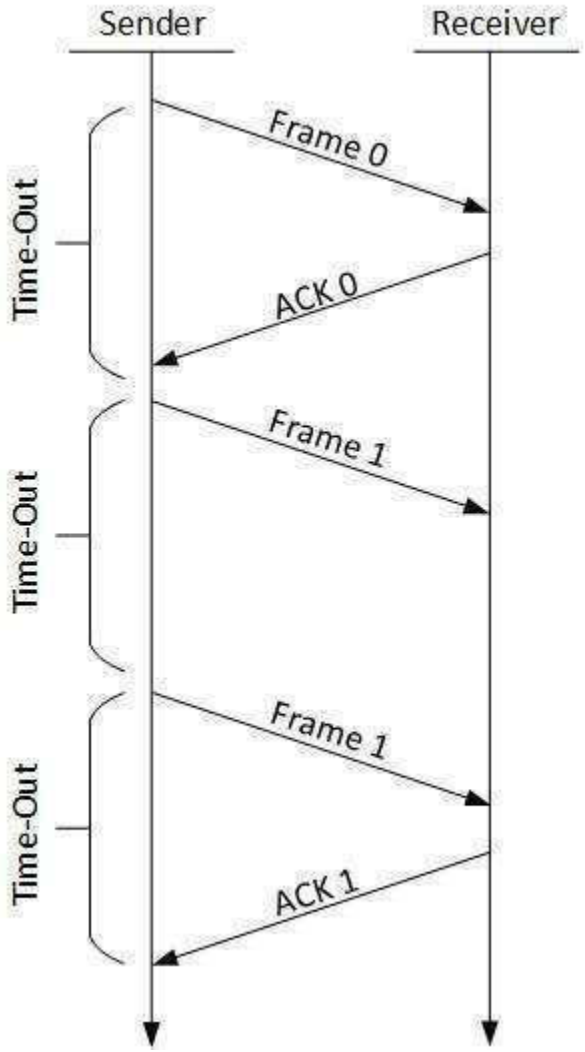
Retransmission: The sender maintains a clock and sets a timeout period. If an acknowledgement of a data-frame previously transmitted does not arrive before the timeout the sender retransmits the frame, thinking that the frame or it's acknowledgement is lost in transit.

There are three types of techniques available which Data-link layer may deploy to control the errors by Automatic Repeat Requests (ARQ):

Stop-and-wait ARQ

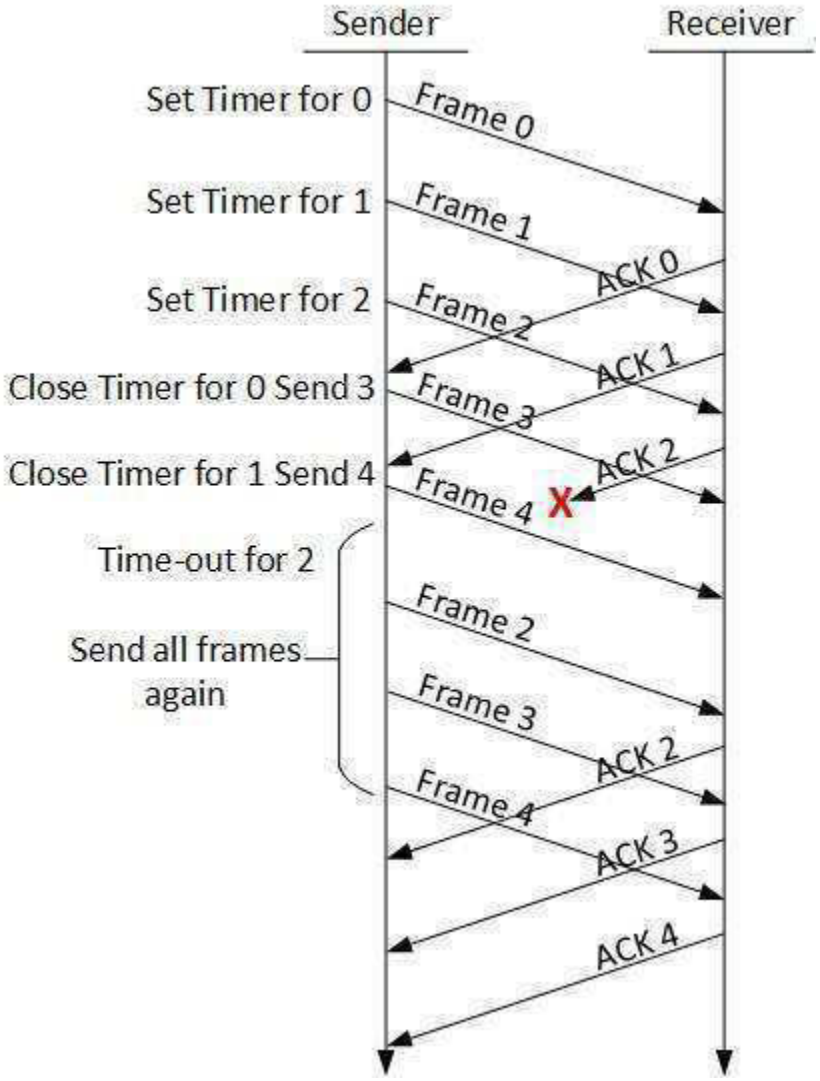
The following transition may occur in Stop-and-Wait ARQ :The sender maintains a timeout counter. When a frame is sent, the sender starts the timeout counter.

If acknowledgement of frame comes in time, the sender transmits the next frame in queue.If acknowledgement does not come in time, the sender assumes that either the frame or its acknowledgement is lost in transit. Sender retransmits the frame and starts the timeout counter.If a negative acknowledgement is received, the sender retransmits the frame.



Go-Back-N ARQ

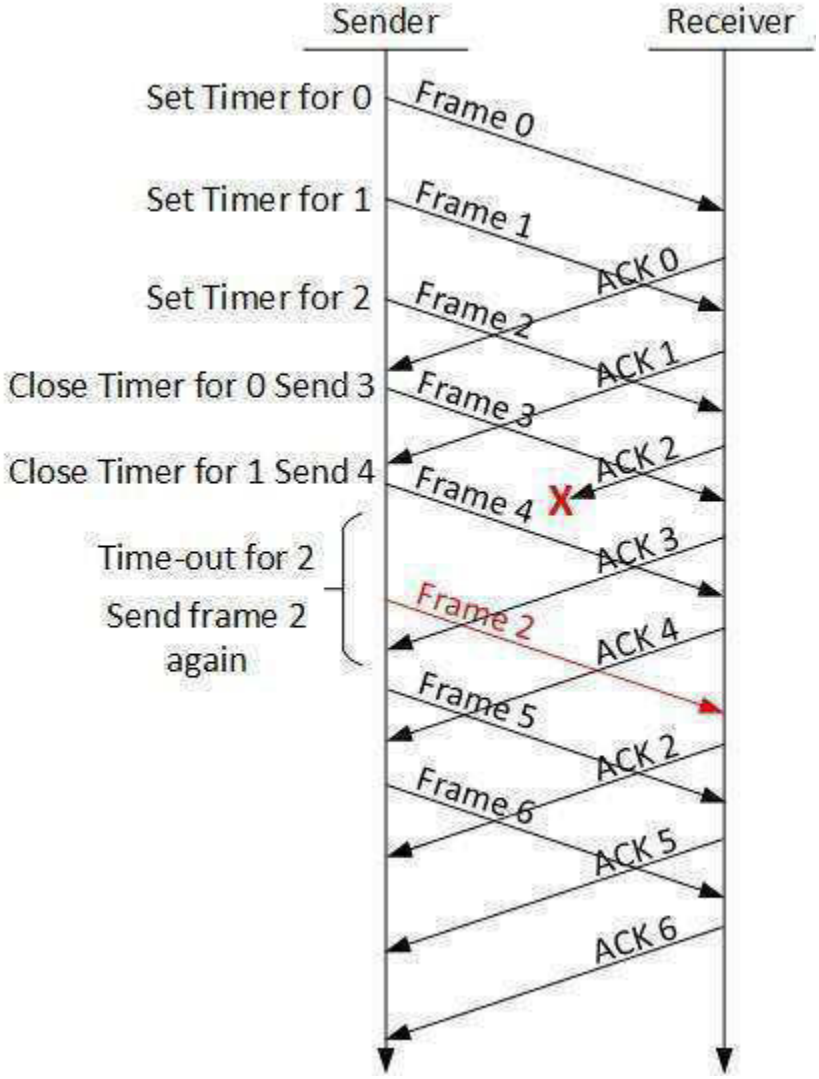
Stop and wait ARQ mechanism does not utilize the resources at their best. When the acknowledgement is received, the sender sits idle and does nothing. In Go-Back-N ARQ method, both sender and receiver maintain a window.



The sending-window size enables the sender to send multiple frames without receiving the acknowledgement of the previous ones. The receiving-window enables the receiver to receive multiple frames and acknowledge them. The receiver keeps track of incoming frame's sequence number.When the sender sends all the frames in window, it checks up to what sequence number it has received positive acknowledgement. If all frames are positively acknowledged, the sender sends next set of frames. If sender finds that it has received NACK or has not receive any ACK for a particular frame, it retransmits all the frames after which it does not receive any positive ACK.

Selective Repeat ARQ

In Go-back-N ARQ, it is assumed that the receiver does not have any buffer space for its window size and has to process each frame as it comes. This enforces the sender to retransmit all the frames which are not acknowledged.



In Selective-Repeat ARQ, the receiver while keeping track of sequence numbers, buffers the frames in memory and sends NACK for only frame which is missing or damaged.The sender in this case, sends only packet for which NACK is received

2.5 ALOHA Protocols

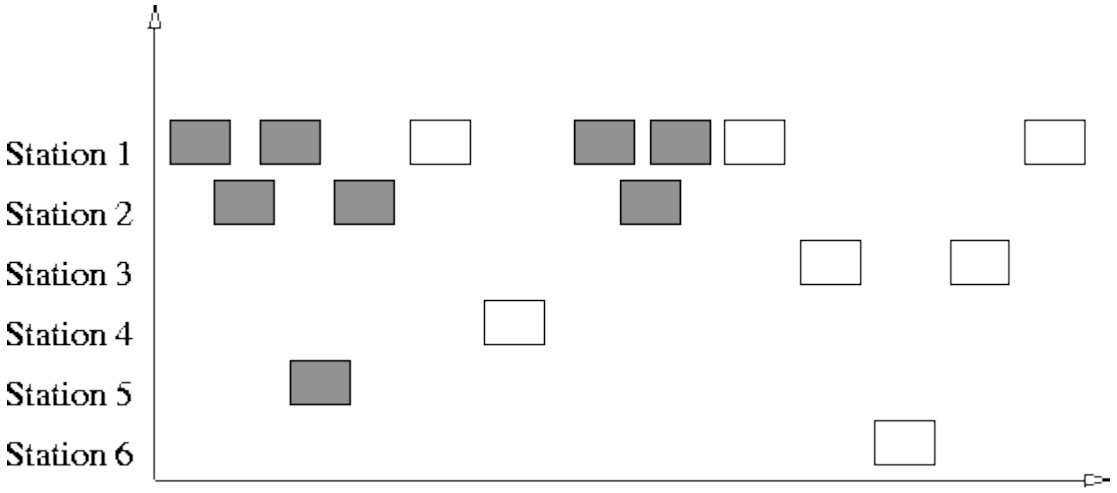
History

The Aloha protocol was designed as part of a project at the University of Hawaii. It provided data transmission between computers on several of the Hawaiian Islands using radio transmissions.

- Communications was typically between remote stations and a central sited named Menehune or vice versa.
- All message to the Menehune were sent using the same frequency.
- When it received a message intact, the Menehune would broadcast an ack on a distinct outgoing frequency.
- The outgoing frequency was also used for messages from the central site to remote computers.
- All stations listened for message on this second frequency.

Pure Aloha

Pure Aloha is an unslotted, fully-decentralized protocol. It is extremely simple and trivial to implement. The ground rule is - "when you want to talk, just talk!". So, a node which wants to transmits will go ahead and send the packet on its broadcast channel, with no consideration whatsoever as to anybody else is transmitting or not.



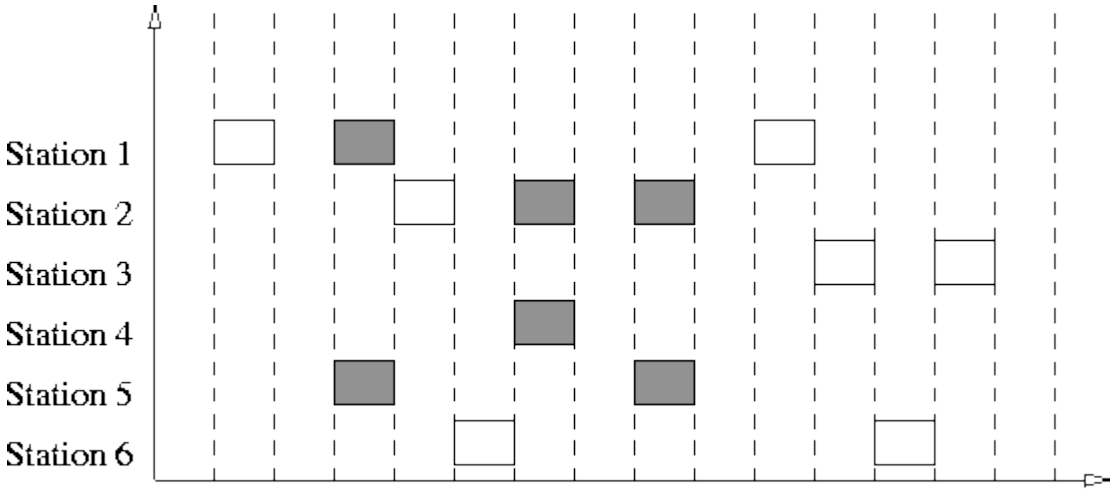
One serious drawback here is that, you dont know whether what you are sending has been received properly or not (so as to say, "whether you've been heard and understood?"). To resolve this, in Pure Aloha, when one node finishes speaking, it

expects an acknowledgement in a finite amount of time - otherwise it simply retransmits the data. This scheme works well in small networks where the load is not high. But in large, load intensive networks where many nodes may want to transmit at the same time, this scheme fails miserably. This led to the development of Slotted Aloha.

Slotted Aloha

This is quite similar to Pure Aloha, differing only in the way transmissions take place. Instead of transmitting right at demand time, the sender waits for some time. This delay is specified as follows - the timeline is divided into equal slots and then it is required that transmission should take place only at slot boundaries. To be more precise, the slotted-Aloha makes the following assumptions:

- All frames consist of exactly L bits.
- Time is divided into slots of size L/R seconds (i.e., a slot equals the time to transmit one frame).
- Nodes start to transmit frames only at the beginnings of slots.
- The nodes are synchronized so that each node knows when the slots begin.
- If two or more frames collide in a slot, then all the nodes detect the collision event before the slot ends.



In this way, the number of collisions that can possibly take place is reduced by a huge margin. And hence, the performance become much better compared to Pure Aloha. collisions may only take place with nodes that are ready to speak at the

same time. But nevertheless, this is a substantial reduction.

Carrier Sense Mutiple Access Protocols

In both slotted and pure ALOHA, a node's decision to transmit is made independently of the activity of the other nodes attached to the broadcast channel. In particular, a node neither pays attention to whether another node happens to be transmitting when it begins to transmit, nor stops transmitting if another node begins to interfere with its transmission. As humans, we have human protocols that allow allows us to not only behave with more civility, but also to decrease the amount of time spent "colliding" with each other in conversation and consequently increasing the amount of data we exchange in our conversations. Specifically, there are two important rules for polite human conversation:

1. **Listen before speaking:** If someone else is speaking, wait until they are done. In the networking world, this is termed carrier sensing - a node listens to the channel before transmitting. If a frame from another node is currently being transmitted into the channel, a node then waits ("backs off") a random amount of time and then again senses the channel. If the channel is sensed to be idle, the node then begins frame transmission. Otherwise, the node waits another random amount of time and repeats this process.
2. **If someone else begins talking at the same time, stop talking.** In the networking world, this is termed collision detection - a transmitting node listens to the channel while it is transmitting. If it detects that another node is transmitting an interfering frame, it stops transmitting and uses some protocol to determine when it should next attempt to transmit.

It is evident that the end-to-end channel propagation delay of a broadcast channel - the time it takes for a signal to propagate from one of the the channel to another - will play a crucial role in determining its performance. The longer this propagation delay, the larger the chance that a carrier-sensing node is not yet able to sense a transmission that has already begun at another node in the network.

CSMA- Carrier Sense Multiple Access

This is the simplest version CSMA protocol as described above. It does not specify any collision detection or handling. So collisions might and WILL occur and clearly then, this is not a very good protocol for large, load intensive networks.

So, we need an improvement over CSMA - this led to the development of CSMA/CD.

CSMA/CD- CSMA with Collision Detection

In this protocol, while transmitting the data, the sender simultaneously tries to receive it. So, as soon as it detects a collision (it doesn't receive its own data) it stops transmitting. Thereafter, the node waits for some time interval before attempting to transmit again. Simply put, **"listen while you talk"**. But, how long should one wait for the carrier to be freed? There are three schemes to handle this:

1. **1-Persistent:** In this scheme, transmission proceeds immediately if the carrier is idle. However, if the carrier is busy, then sender continues to sense the carrier until it becomes idle. The main problem here is that, if more than one transmitters are ready to send, a collision is GUARANTEED!!
2. **Non-Persistent:** In this scheme, the broadcast channel is not monitored continuously. The sender polls it at random time intervals and transmits whenever the carrier is idle. This decreases the probability of collisions. But, it is not efficient in a low load situation, where number of collisions are anyway small. The problems it entails are:
    - If back-off time is too long, the idle time of carrier is wasted in some sense
    - It may result in long access delays
3. **p-Persistent:** Even if a sender finds the carrier to be idle, it uses a probabilistic distribution to determine whether to transmit or not. Put simply, "toss a coin to decide". If the carrier is idle, then transmission takes place with a probability p and the sender waits with a probability 1-p. This scheme is a good trade off between the Non-persistent and 1-persistent schemes. So, for low load situations, p is high (example: 1-persistent); and for high load situations, p may be lower. Clearly, the value of p plays an important role in determining the performance of this protocol. Also the same p is likely to provide different performance at different loads.

CSMA/CD doesn't work in some wireless scenarios called **"hidden node"** problems. Consider a situation, where there are 3 nodes - A, B and C communicating with each other using a wireless protocol. Morover, B can communicate with both A and C, but A and C lie outside each other's range and hence can't communicate directly with each other. Now, suppose both A and C want to communicate with B simultaneously. They both will sense the carrier to be idle and hence will begin transmission, and even if there is a collision, neither A nor C will ever detect it. B on the other hand will receive 2 packets at the same time and might not be able to understand either of them. To get around this problem, a better version called CSMA/CA was developed, specially for wireless applications.