UNIT-II

Syllabus:

Data Link Layer - Design issues, CRC codes, Elementary Data Link Layer Protocols, sliding window protocol.

Multi Access Protocols - ALOHA, CSMA, Collision free protocols, Ethernet- Physical Layer, Ethernet Mac Sub layer – CSMA/CD with Binary Exponential Back off, Ethernet Performance, Switched, Fast, Gigabit, 10-Gigabit Ethernets, Data link layer switching & use of bridges, learning bridges, spanning tree bridges, repeaters, hubs, bridges, switches, routers and gateways.

Design Issues

Services Provided to the Network Layer

Framing

Physical Addressing

Error Control

Flow Control

Access Control

1. Services Provided to the Network Layer

The function of the data link layer is to provide services to the network layer. The principal service is transferring data from the network layer on the source machine to the network layer on the destination machine.

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

Unacknowledged Connectionless service

Acknowledged Connectionless service

Acknowledged Connection-Oriented service

Unacknowledged connectionless service

No acks, no connection

Error recovery up to higher layers

For low error-rate links or voice traffic

Acknowledged connectionless service

Acks improve reliability

For unreliable channels. E.g.: Wireless Systems

Acknowledged connection-oriented service

Equivalent of reliable bit-stream

Connection establishment

Packets Delivered In-Order

Connection Release

Inter-Router Traffic

Framing

To provide service to the network layer, the data link layer must use the service provided to it by the physical layer. What the physical layer does is accept a raw bit stream and attempt to deliver it to the destination. This bit stream is not guaranteed to be error free. The number of bits received may be less than, equal to, or more than the number of bits transmitted, and they may have different values. It is up to the data link layer to detect and, if necessary, correct errors. The usual approach is for the data link layer to break the bit stream up into discrete frames and compute the checksum for each frame. When a frame arrives at the destination, the checksum is

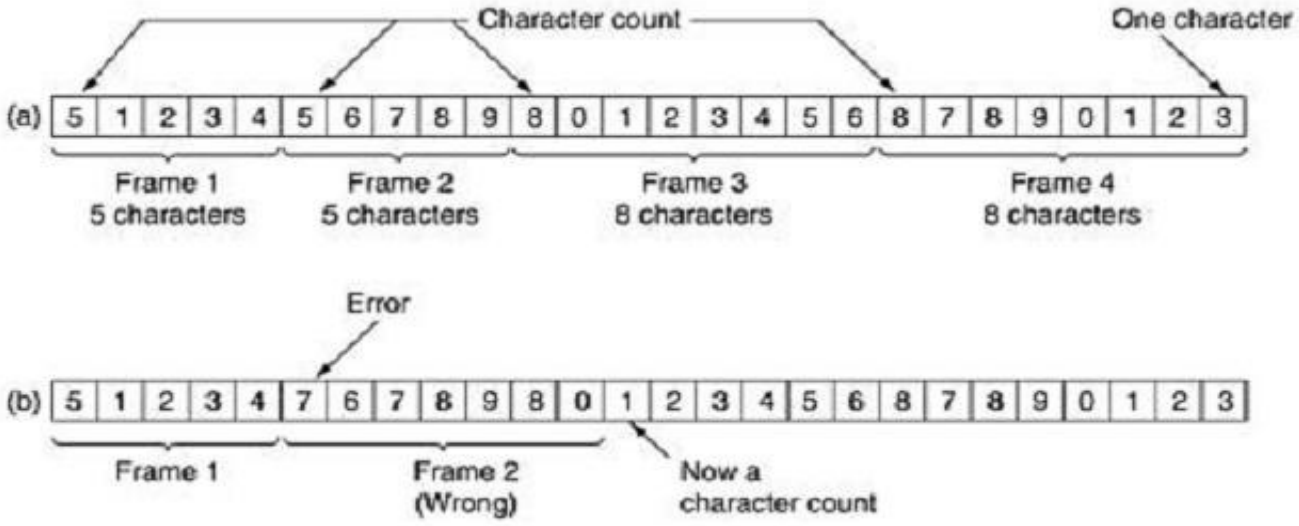
recomputed. If the newly computed checksum is different from the one contained in the frame, the data link layer knows that an error has occurred and takes steps to deal with it (e.g., discarding the bad frame and possibly also sending back an error report).

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. Since it is too risky to count on timing to mark the start and end of each frame, other methods have been devised. We will look at four methods:

1. Character count.
2. Flag bytes with byte stuffing.
3. Starting and ending flags, with bit stuffing.
4. Physical layer coding violations.

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.3.1(a) for four frames of sizes 5, 5, 8, and 8 characters, respectively

Fig.3.1 A character stream. (a) Without errors. (b) With one error.



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

The second framing method gets around the problem of resynchronization after an error by having each frame start and end with special bytes. In the past, the starting and ending bytes were different, but in recent years most protocols have used the same byte, called a flag byte, as both the starting and ending delimiter, as shown in Fig. 3.2(a) as FLAG. In this way, if the receiver ever loses synchronization, it can just search for the flag byte to find the end of the current frame. Two consecutive flag bytes indicate the end of one frame and start of the next one.

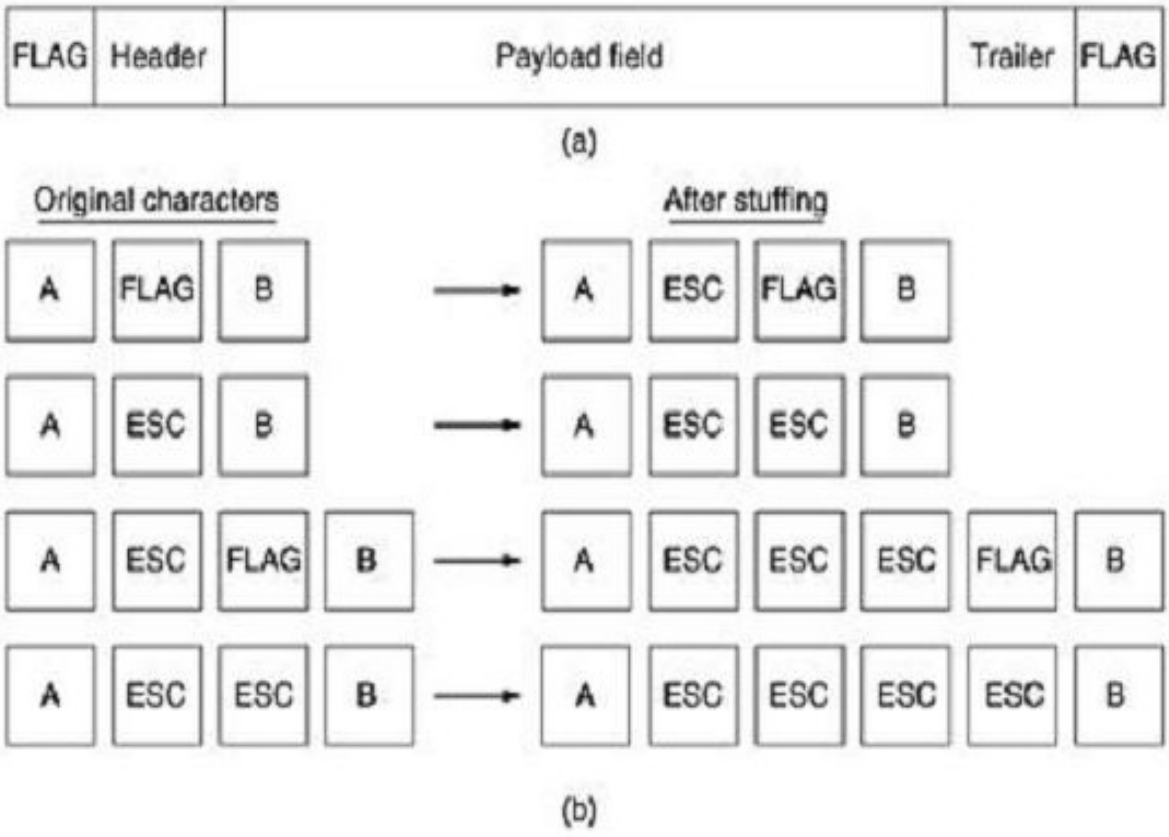


Fig. 3.2 (a) A frame delimited by flag bytes (b) Four examples of byte sequences before and after byte stuffing.

A serious problem occurs with this method when binary data, such as object programs or floating-point numbers, are being transmitted. It may easily happen that the flag byte's bit pattern occurs in the data. This situation will usually interfere with the framing. One way to solve this problem is to have the sender's data link layer insert a special escape byte (ESC) just before each "accidental" flag byte in the data. The data link layer on the receiving end removes the escape byte before the data are given to the network layer. This technique is called byte stuffing or character stuffing. Thus, a framing flag byte can be distinguished from one in the data by the absence or presence of an escape byte before it.

Of course, the next question is: What happens if an escape byte occurs in the middle of the data? The answer is that it, too, is stuffed with an escape byte. Thus, any single escape byte is part of an escape sequence, whereas a doubled one indicates that a single escape occurred naturally in the data. Some examples are shown in Fig. 3.3(b). In all cases, the byte sequence delivered after de stuffing is exactly the same as the original byte sequence.

The byte-stuffing scheme depicted in Fig. 3.3 is a slight simplification of the one used in the PPP protocol that most home computers use to communicate with their Internet service provider.

A major disadvantage of using this framing method is that it is closely tied to the use of 8-bit characters. Not all character codes use 8-bit characters. For example UNICODE uses 16-bit characters, As networks developed, the disadvantages of embedding the character code length in the framing mechanism became more and more obvious, so a new technique had to be developed to allow arbitrary sized characters.

The new technique allows data frames to contain an arbitrary number of bits and allows character codes with an arbitrary number of bits per character. It works like this. Each frame begins and ends with a special bit pattern, 01111110 (in fact, a flag byte). Whenever the sender's data link layer encounters five consecutive 1s in the data, it automatically stuffs a 0 bit into the outgoing bit stream. This bit stuffing is analogous to byte stuffing, in which an escape byte is stuffed into the outgoing character stream before a flag byte in the data.

When the receiver sees five consecutive incoming 1 bits, followed by a 0 bit, it automatically de stuffs (i.e., deletes) the 0 bit. Just as byte stuffing is completely transparent to the network layer

in both computers, so is bit stuffing. If the user data contain the flag pattern, 01111110, this flag is transmitted as 011111010 but stored in the receiver's memory as 01111110.

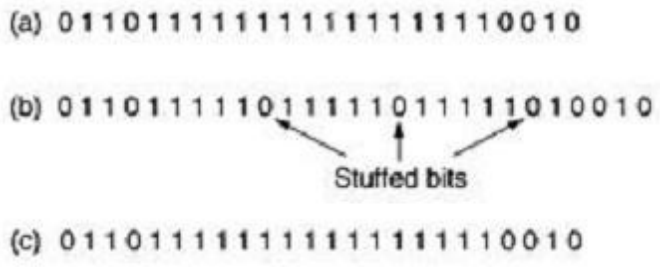


Figure 3.3 Bit stuffing. (a) The original data. (b) The data as they appear on the line. (c) The data as they are stored in the receiver's memory after destuffing.

With bit stuffing, the boundary between two frames can be unambiguously recognized by the flag pattern. Thus, if the receiver loses track of where it is, all it has to do is scan the input for flag sequences, since they can only occur at frame boundaries and never within the data. The last method of framing is only applicable to networks in which the encoding on the physical medium contains some redundancy. For example, some LANs encode 1 bit of data by using 2 physical bits. Normally, a 1 bit is a high-low pair and a 0 bit is a low-high pair. The scheme means that every data bit has a transition in the middle, making it easy for the receiver to locate the bit

boundaries. The combinations high-high and low-low are not used for data but are used for delimiting frames in some protocols.

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

3.1 Error correction and detection at the data link layer.

3.1.1 Error-Correcting Codes:

Network designers have developed two basic strategies for dealing with errors. One way is to include enough redundant information along with each block of data sent, to enable the receiver to deduce what the transmitted data must have been. The other way is to include only enough redundancy to allow the receiver to deduce that an error occurred, but not which error, and have it request a retransmission. The former strategy uses error-correcting codes and the latter uses error-detecting codes. The use of error-correcting codes is often referred to as forward error correction.

Each of these techniques occupies a different ecological niche. On channels that are highly reliable, such as fiber, it is cheaper to use an error detecting code and just retransmit the occasional block found to be faulty. However, on channels such as wireless links that make many errors, it is better to add enough redundancy to each block for the receiver to be able to figure out what the original block was, rather than relying on a retransmission, which itself may be in error.

To understand how errors can be handled, it is necessary to look closely at what an error really is. Normally, a frame consists of m data (i.e., message) bits and r redundant, or check, bits. Let the total length be n (i.e., n = m + r). An n-bit unit containing data and check bits is often referred to as an n-bit codeword.

Given any two code words, say, 10001001 and 10110001, it is possible to determine how many corresponding bits differ. In this case, 3 bits differ. To determine how many bits differ, just exclusive OR the two code words and count the number of 1 bits in the result, for example:

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

In most data transmission applications, all 2m possible data messages are legal, but due to the way the check bits are computed, not all of the 2n possible codewords are used. Given the algorithm for computing the check bits, it is possible to construct a complete list of the legal codewords, and from this list find the two codewords whose Hamming distance is minimum. This distance is the Hamming distance of the complete code. The error-detecting and error correcting properties of a code depend on its Hamming distance. To detect d errors, you need a distance d + 1 code because with such a code there is no way that d single-bit errors can change a valid codeword into another valid codeword. When the receiver sees an invalid codeword, it can tell that a transmission error has occurred. Similarly, to correct d errors, you need a distance 2d + 1 code because that way the legal codewords are so far apart that even with d changes, the original codeword is still closer than any other codeword, so it can be uniquely determined. As a simple example of an error-detecting code, consider a code in which a single parity b appended to the data. The parity bit is chosen so that the number of 1 bits in the codeword is even (or odd). For example, when 1011010 is sent in even parity, a bit is added to the end to make it 10110100. With odd parity 1011010 becomes 10110101. A code with a single parity bit has a distance 2, since any single-bit error produces a codeword with the wrong parity. It can be used to detect single errors.

As a simple example of an error-correcting code, consider a code with only four valid codewords: 0000000000, 0000011111, 1111100000, and 1111111111

This code has a distance 5, which means that it can correct double errors. If the codeword 0000000111 arrives, the receiver knows that the original must have been 0000011111. If, however, a triple error changes 0000000000 into 0000000111, the error will not be corrected properly.

Imagine that we want to design a code with m message bits and r check bits that will allow all single errors to be corrected. Each of the 2m legal messages has n illegal codewords at a distance 1 from it. These are formed by systematically inverting each of the n bits in the n-bit codeword formed from it. Thus, each of the 2m legal messages requires n + 1 bit patterns dedicated to it. Since the total number of bit patterns is 2n, we must have $(n + 1)2m \leq 2n$.

Using n = m + r, this requirement becomes (m + r + 1) ≤ 2r. Given m, this puts a lower limit on the number of check bits needed to correct single errors. This theoretical lower limit can, in fact, be achieved using a method due to Hamming (1950).

The bits of the codeword are numbered consecutively, starting with bit 1 at the left end, bit 2 to its immediate right, and so on. The bits that are powers of 2 (1, 2, 4, 8, 16, etc.) are check bits. The rest (3, 5, 6, 7, 9, etc.) are filled up with the m data bits. Each check bit forces the parity of some collection of bits, including itself, to be even (or odd). A bit may be included in several parity computations. To see which check bits the data bit in position k contributes to, rewrite k as a sum of powers of 2.For example, 11 = 1 + 2 + 8 and 29 = 1 + 4 + 8 + 16. A bit is checked by just those check bits occurring in its expansion (e.g., bit 11 is checked by bits 1, 2, and 8). When a codeword arrives, the receiver initializes a counter to zero. It then examines each check bit, k (k = 1, 2, 4, 8 ...), to see if it has the correct parity. If not, the receiver adds k to the counter. If the counter is zero after all the check bits have been examined (i.e., if they were all correct), the codeword is accepted as valid. If the counter is nonzero, it contains the number of the incorrect bit. For example, if check bits 1, 2, and 8 are in error, the inverted bit is 11, because it is the only one checked by bits 1, 2, and 8. Figure 4.1 shows some 7-bit ASCII characters encoded as 11-bit codewords using a Hamming code. Remember that the data are found in bit positions 3, 5, 6, 7, 9, 10, and 11.

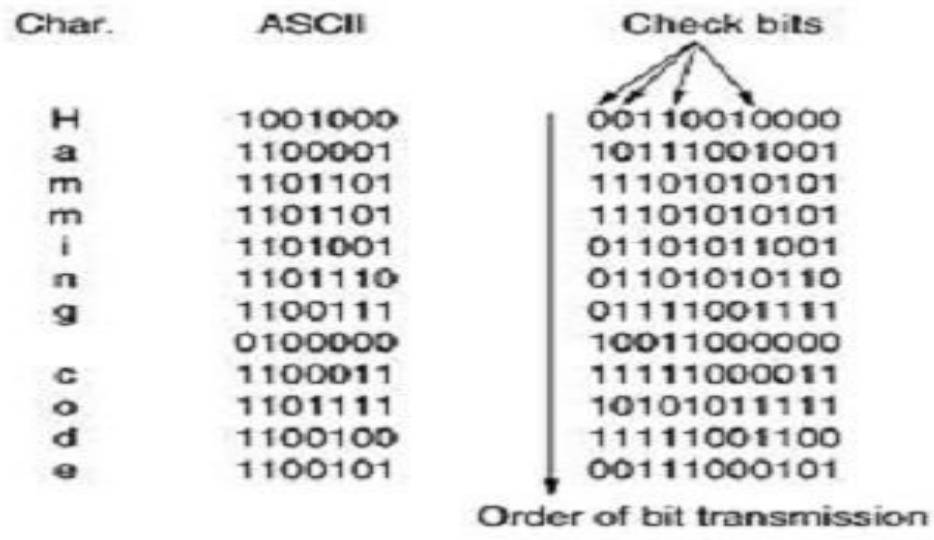


Fig.4.1. Use of a Hamming code to correct burst errors

Hamming codes can only correct single errors. However, there is a trick that can be used to permit Hamming codes to correct burst errors. A sequence of k consecutive code words is arranged as a matrix, one codeword per row. Normally, the data would be transmitted one codeword at a time, from left to right. To correct burst errors, the data should be transmitted one column at a time, starting with the leftmost column. When all k bits have been sent, the second column is sent, and so on, as indicated in Fig.4.1. When the frame arrives at the receiver, the matrix is reconstructed, one column at a time. If a burst error of length k occurs, at most 1 bit in each of the k codewords will have been affected, but the Hamming code can correct one error per codeword, so the entire block can be restored. This method uses kr check bits to make blocks of km data bits immune to a single burst error of length k or less.

3.1.2 Error-Detecting Codes:

Error-correcting codes are widely used on wireless links, which are notoriously noisy and error prone when compared to copper wire or optical fibers. Without error-correcting codes, it would be hard to get anything through. However, over copper wire or fiber, the error rate is much lower, so error detection and retransmission is usually more efficient there for dealing with the occasional error. As a simple example, consider a channel on which errors are isolated and the error rate is 10-6 per bit. Let the block size be 1000 bits. To provide error correction for 1000-bit blocks, 10 check bits are needed; a megabit of data would require 10,000 check bits. To merely detect a block with a single 1-bit error, one parity bit per block will suffice. Once every 1000 blocks, an extra block (1001 bits) will have to be transmitted. The total overhead for the error detection + retransmission method is only 2001 bits per megabit of data, versus 10,000 bits for a Hamming code.

If a single parity bit is added to a block and the block is badly garbled by a long burst error, the probability that the error will be detected is only 0.5, which is hardly acceptable.

The odds can be improved considerably if each block to be sent is regarded as rectangular matrix n bits wide and k bits high, as described above. A parity bit is computed separately for each column and affixed to the matrix as the last row. The matrix is then transmitted one row at a time. When the block arrives, the receiver checks all the parity bits. If any one of them is wrong, the receiver requests a retransmission of the block. Additional retransmissions are requested as needed until an entire block is received without any parity errors. This method can detect a single burst of length n, since only 1 bit per column will be changed. A burst of length n + 1 will pass

undetected, however, if the first bit is inverted, the last bit is inverted, and all the other bits are correct. (A burst error does not imply that all the bits are wrong; it just implies that at least the first and last are wrong.) If the block is badly garbled by a long burst or by multiple shorter bursts, the probability that any of the n columns will have the correct parity, by accident, is 0.5, so the probability of a bad block being accepted when it should not be is 2-n. Although the above cheme may sometimes be adequate, in practice, another method is in widespread use: the olynomial code, also known as a CRC (Cyclic Redundancy Check).

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 x0. Such a polynomial is said to be of degree k - 1. The highorder (leftmost) bit is the coefficient of xk-1; the next bit is the coefficient of xk-2, and so on. For example, 110001 has 6 bits and thus represent a six-term polynomial with coefficients 1, 1, 0, 0, 0, and 1: x5 + x4 + x0.

Polynomial arithmetic is done modulo 2, according to the rules of algebraic field theory. There are no carries for addition or borrows for subtraction. Both addition and subtraction are identical to exclusive OR. For example: Long division is carried out the same way as it is in binary except that the subtraction is done modulo 2, as above. A divisor is said "to go into" a dividend if the dividend has as many bits as the divisor. 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 checksum 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 checksum 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 checksum is as follows:

1. 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).

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

3. Subtract the remainder (which is always r or fewer bits) from the bit string corresponding to xr M(x) using modulo 2 subtractions. The result is the checksummed frame to be transmitted. Call its polynomial T(x).

Figure illustrates the calculation for a frame 1101011011 using the generator G(x) = x4 + x+ 1.

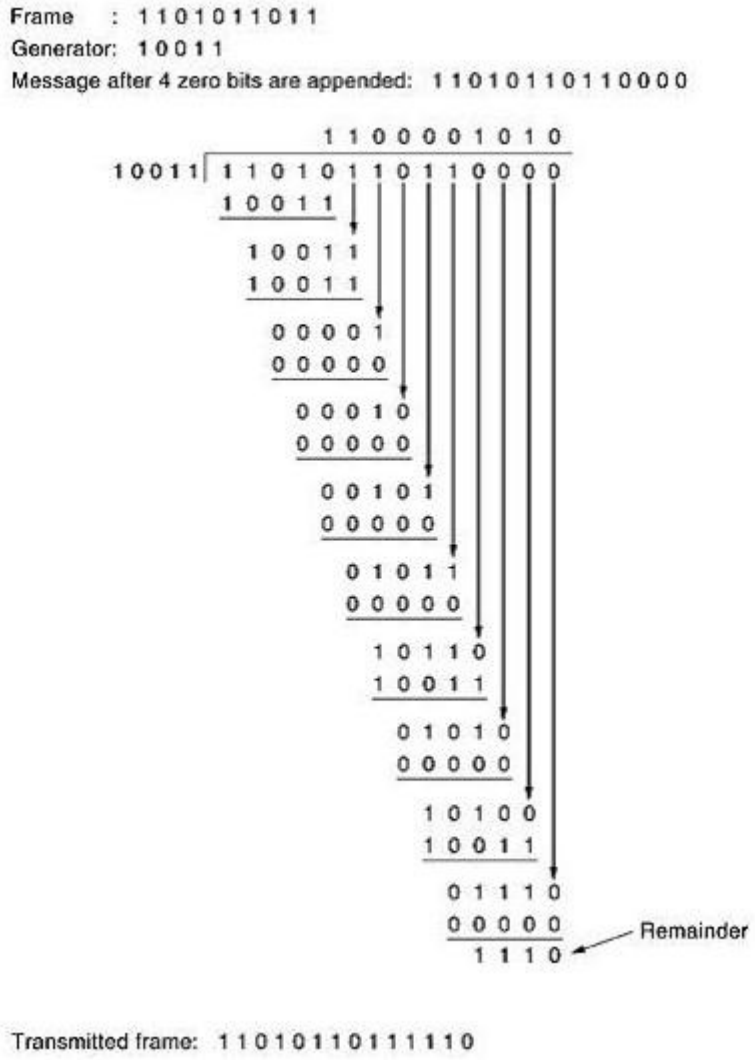


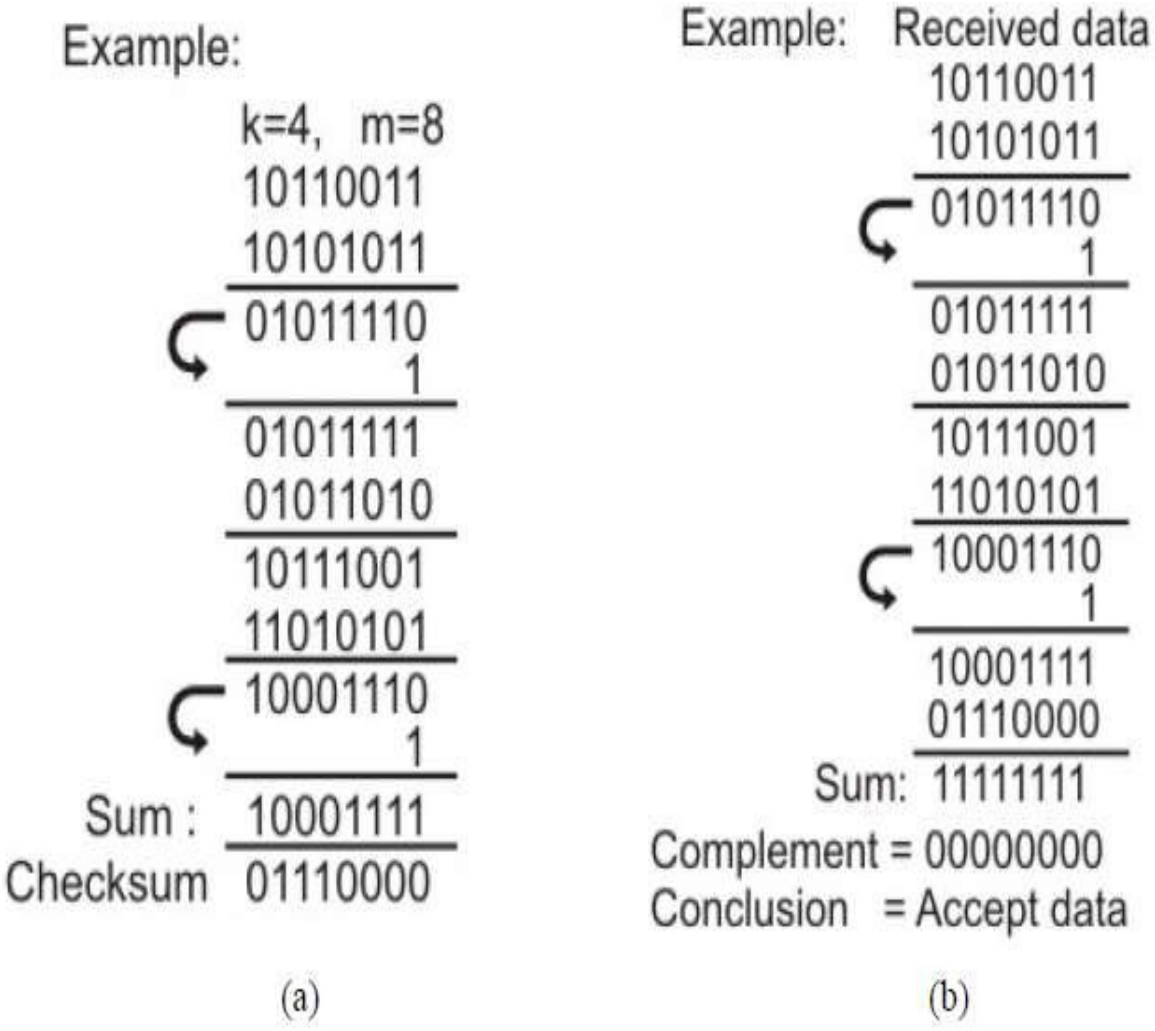
Fig.5.1. Calculation of the polynomial code checksum

Checksum

Checksum is the error detection scheme used in IP, TCP & UDP.

Here, 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

Sender:	Receiver:
• data is divided into k sections each n bits long	• data is divided into k sections each n bits long
• all sections are added using 1-s complement to get the sum	• all sections are added using 1-s complement to get the sum
• the sum is bit-wise complemented and becomes the checksum	• the sum is bit-wise complemented
• the checksum is sent with the data	• if the result is zero, the data is accepted, otherwise it is rejected



(a) Sender's end for the calculation of the checksum. (b) Receiving end for checking the checksum

3.2 Elementary Data Link Layer Protocols

3.2.1 An Unrestricted Simplex Protocol:

As an initial example we will consider a protocol that is as simple as it can be. Data are transmitted in one direction only. Both the transmitting and receiving network layers are always ready. Processing time can be ignored. Infinite buffer space is available. And best of all, the communication channel between the data link layers never damages or loses frames. This thoroughly unrealistic protocol, which we will nickname "utopia" .

The protocol consists of two distinct procedures, a sender and a receiver. The sender runs in the data link layer of the source machine, and the receiver runs in the data link layer of the destination machine. No sequence numbers or acknowledgements are used here, so MAX_SEQ is not needed. The only event type possible is frame_arrival (i.e., the arrival of an undamaged frame).

The sender is in an infinite while loop just pumping data out onto the line as fast as it can. The body of the loop consists of three actions: go fetch a packet from the (always obliging) network layer, construct an outbound frame using the variable s, and send the frame on its way. Only the info field of the frame is used by this protocol, because the other fields have to do with error and flow control and there are no errors or flow control restrictions here. The receiver is equally simple. Initially, it waits for something to happen, the only possibility being the arrival of an undamaged frame. Eventually, the frame arrives and the procedure wait_for_event returns, with event set to frame_arrival (which is ignored anyway). The call to from_physical_layer removes the newly arrived frame from the hardware buffer and puts it in the variable r, where the receiver code can get at it. Finally, the data portion is passed on to the network layer, and the data link layer settles back to wait for the next frame, effectively suspending itself until the frame arrives.

3.2.2 A Simplex Stop-and-Wait Protocol:

The main problem we have to deal with here is how to prevent the sender from flooding the receiver with data faster than the latter is able to process them. In essence, if the receiver requires a time Δt to execute from_physical_layer plus to_network_layer, the sender must transmit at an average rate less than one frame per time Δt. Moreover, if we assume that no automatic buffering and queuing are done within the receiver's hardware, the sender must never transmit a new frame until the old one has been fetched by from_physical_layer, lest the new one overwrite the old

one. In certain restricted circumstances (e.g., synchronous transmission and a receiving data link layer fully dedicated to processing the one input line), it might be possible for the sender to simply insert a delay into protocol 1 to slow it down sufficiently to keep from swamping the receiver. However, more usually, each data link layer will have several lines to attend to, and the time interval between a frame arriving and its being processed may vary considerably. If the network designers can calculate the worst-case behavior of the receiver, they can program the sender to transmit so slowly that even if every frame suffers the maximum delay, there will be no overruns. The trouble with this approach is that it is too conservative. It leads to a bandwidth utilization that is far below the optimum, unless the best and worst cases are almost the same (i.e., the variation in the data link layer's reaction time is small).

A more general solution to this dilemma is to have the receiver provide feedback to the sender. After having passed a packet to its network layer, the receiver sends a little dummy frame back to the sender which, in effect, gives the sender permission to transmit the next frame. After having sent a frame, the sender is required by the protocol to bide its time until the little dummy (i.e., acknowledgement) frame arrives. Using feedback from the receiver to let the sender know when it may send more data is an example of the flow control mentioned earlier.

Protocols in which the sender sends one frame and then waits for an acknowledgement before proceeding are called stop-and-wait.

3.2.3 A Simplex Protocol for a Noisy Channel:

Protocols in which the sender waits for a positive acknowledgement before advancing to the next data item are often called PAR (Positive Acknowledgement with Retransmission) or ARQ (Automatic Repeat reQuest). Like protocol 2, this one also transmits data only in one direction.

4.1 Sliding Window Protocols:

In the previous protocols, data frames were transmitted in one direction only. In most practical situations, there is a need for transmitting data in both directions. One way of achieving fullduplex data transmission is to have two separate communication channels and use each one for simplex data traffic (in different directions). If this is done, we have two separate physical circuits, each with a "forward" channel (for data) and a "reverse" channel (for acknowledgements). In both cases the bandwidth of the reverse channel is almost entirely wasted. In effect, the user is paying for two circuits but using only the capacity of one. A better idea is to use the same circuit for data in both directions. After all, in protocols 2 and 3 it was already being used to transmit frames both ways, and the reverse channel has the same capacity as the forward channel. In this model the data frames from A to B are intermixed with the acknowledgement frames from A to B. By looking at the kind field in the header of an incoming frame, the receiver can tell whether the frame is data or acknowledgement.

Although interleaving data and control frames on the same circuit is an improvement over having two separate physical circuits, yet another improvement is possible. When a data frame arrives, instead of immediately sending a separate control frame, the receiver restrains itself and waits until the network layer passes it the next packet. The acknowledgement is attached to the outgoing data frame (using the ack field in the frame header). In effect, the acknowledgement gets a free ride on the next outgoing data frame. The technique of temporarily delaying outgoing acknowledgements so that they can be hooked onto the next outgoing data frame is known as piggybacking. The principal advantage of using piggybacking over having distinct acknowledgement frames is a better use of the available channel bandwidth. The ack field in the frame header costs only a few bits, whereas a separate frame would need a header, the acknowledgement, and a checksum. In addition, fewer frames sent means fewer "frame arrival" interrupts, and perhaps fewer buffers in the receiver, depending on how the receiver's software is organized. In the next protocol to be examined, the piggyback field costs only 1 bit in

the frame header. It rarely costs more than a few bits.

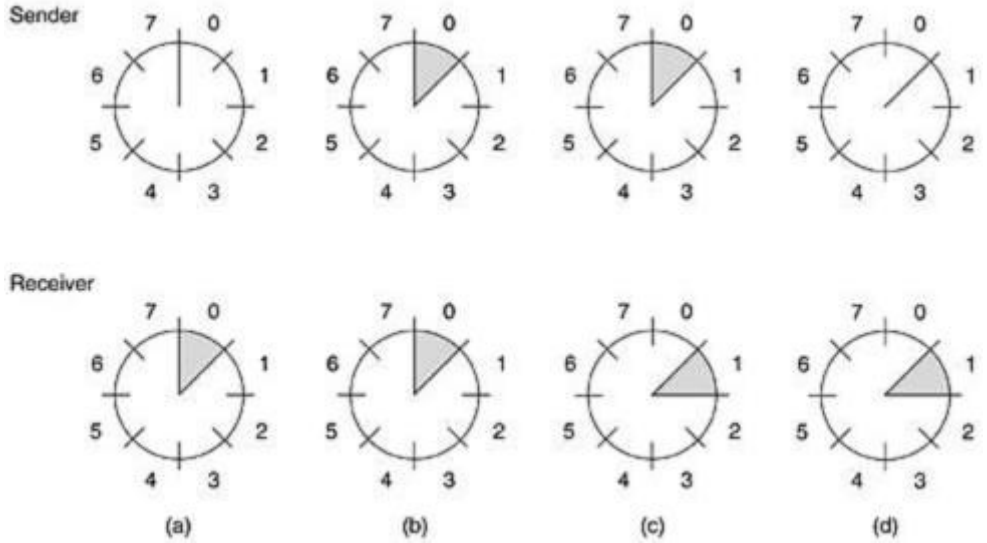


Fig.8.A sliding window of size 1, with a 3-bit sequence number (a) Initially (b) After the first frame has been sent (c) After the first frame has been received (d) After the first acknowledgement has been received.

Since frames currently within the sender's window may ultimately be lost or damaged in transit, the sender must keep all these frames in its memory for possible retransmission. Thus, if the maximum window size is n, the sender needs n buffers to hold the unacknowledged frames. If the window ever grows to its maximum size, the sending data link layer must forcibly shut off the network layer until another buffer becomes free. The receiving data link layer's window corresponds to the frames it may accept. Any frame falling outside the window is discarded without comment. When a frame whose sequence number is equal to the lower edge of the window is received, it is passed to the network layer, an acknowledgement is generated, and the window is rotated by one. Unlike the sender's window, the receiver's window always remains at its initial size. Note that a window size of 1 means that the data link layer only accepts frames in order, but for larger windows this is not so. The network layer, in contrast, is always fed data in the proper order, regardless of the data link layer's window size. Figure 8 shows an example with a maximum window size of 1. Initially, no frame are outstanding, so the lower and upper edges of the sender’s window are equal, but as time goes on, the situation progresses as shown.

4.1.1 A One-Bit Sliding Window Protocol:

Before tackling the general case, let us first examine a sliding window protocol with a maximum window size of 1. Such a protocol uses stop-and-wait since the sender transmits a frame and waits for its acknowledgement before sending the next one. Figure 9.1 depicts such a protocol. Like the others, it starts out by defining some variables. Next_frame_to_send tells which frame the sender is trying to send. Similarly, frame_expected tells which frame the receiver is expecting. In both cases, 0 and 1 are the only possibilities.

Under normal circumstances, one of the two data link layers goes first and transmits the first frame. In other words, only one of the data link layer programs should contain the to_physical_layer and start_timer procedure calls outside the main loop. In the event that both data link layers start off simultaneously, a peculiar situation arises, as discussed later. 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. Now let us examine protocol 4 to see how resilient it is to pathological scenarios. Assume that computer A is trying to send its frame 0 to computer B and that B is trying to send its frame 0 to A. Suppose that A sends frame to B, but A's timeout interval is a little too short. Consequently, A may time out repeatedly, sending a series of identical frames, all with seq = 0 and ack = 1.

When the first valid frame arrives at computer B, it will be accepted and frame_expected will be set to 1. All the subsequent frames will be rejected because B is now expecting frames with sequence number 1, not 0. Furthermore, since all the duplicates have ack = 1 and B is still waiting for an acknowledgement of 0, B will not fetch a new packet from its network layer. After every rejected duplicate comes in, B sends A a frame containing seq = 0 and ack = 0. Eventually, one of these arrives correctly at A, causing A to begin sending the next packet. No combination of lost frames or premature timeouts can cause the protocol to deliver duplicate packets to either network layer, to skip a packet, or to deadlock.

However, a peculiar situation arises if both sides simultaneously send an initial packet. This synchronization difficulty is illustrated by Fig.9.2. In part (a), the normal operation of the protocol is shown. In (b) the peculiarity is illustrated. If B waits for A's first frame before sending one of its own, the sequence is as shown in (a), and every frame is accepted. However, if A and B simultaneously initiate communication, their first frames cross, and the data link layers then get into situation (b). In (a) each frame arrival brings a new packet for the network layer; there are no duplicates. In (b) half of the frames contain duplicates, even though there are no transmission errors. Similar situations can occur as a result of premature timeouts, even when one side clearly starts first. In fact, if multiple premature timeouts occur, frames may be sent three or more times.

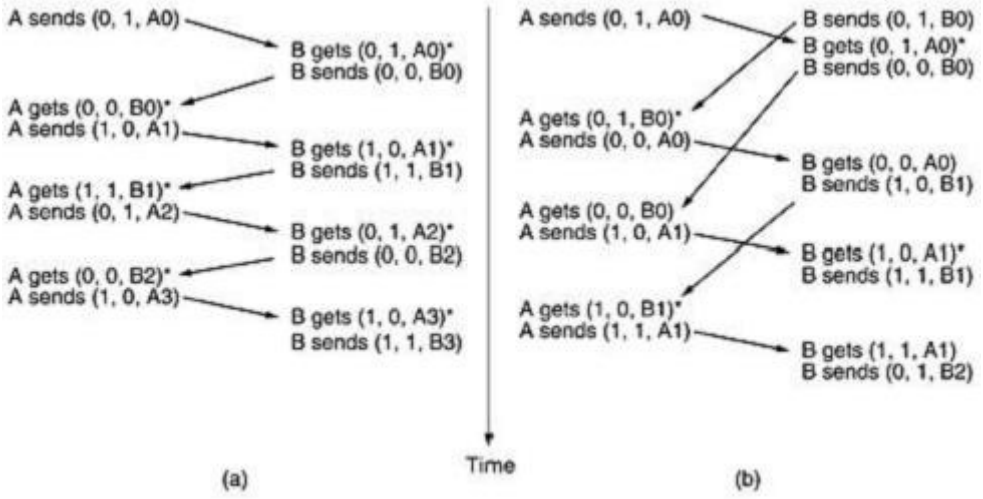


Fig.9.2 Two scenarios for protocol 4 (a) Normal case (b) Abnormal case. The notation is (seq, ack, packet number). An asterisk indicates where a network layer accepts a packet.

4.1.2 Selective repeat:

A Protocol Using Go Back N:

Until now we have made the tacit assumption that the transmission time required for a frame to arrive at the receiver plus the transmission time for the acknowledgement to come back is negligible. Sometimes this assumption is clearly false. In these situations the long round-trip time can have important implications for the efficiency of the bandwidth utilization. As an example, consider a 50-kbps satellite channel with a 500-msec round-trip propagation delay. Let

us imagine trying to use protocol 4 to send 1000-bit frames via the satellite. At t = 0 the sender starts sending the first frame. At t = 20 msec the frame has been completely sent. Not until t = 270 msec has the frame fully arrived at the receiver, and not until t = 520 msec has the acknowledgement arrived back at the sender, under the best of circumstances (no waiting in the receiver and a short acknowledgement frame). This means that the sender was blocked during 500/520 or 96 percent of the time. In other words, only 4 percent of the available bandwidth was used. Clearly, the combination of a long transit time, high bandwidth, and short frame length is disastrous in terms of efficiency. The problem described above can be viewed as a consequence of the rule requiring a sender to wait for an acknowledgement before sending another frame. If we relax that restriction, much better efficiency can be achieved. Basically, the solution lies in allowing the sender to transmit up to w frames before blocking, instead of just 1. With an appropriate choice of w the sender will be able to continuously transmit frames for a time equal to the round-trip transit time without filling up the window. In the example above, w should be at least 26. The sender begins sending frame 0 as before. By the time it has finished sending 26 frames, at t = 520, the acknowledgement for frame 0 will have just arrived. Thereafter, acknowledgements arrive every 20 msec, so the sender always gets permission to continue just when it needs it. At all times, 25 or 26 unacknowledged frames are outstanding. Put in other terms, the sender's maximum window size is 26.

The need for a large window on the sending side occurs whenever the product of bandwidth x round-trip-delay is large. If the bandwidth is high, even for a moderate delay, the sender will exhaust its window quickly unless it has a large window. If the delay is high (e.g., on a geostationary satellite channel), the sender will exhaust its window even for a moderate bandwidth. The product of these two factors basically tells what the capacity of the pipe is, and the sender needs the ability to fill it without stopping in order to operate at peak efficiency. This technique is known as pipelining. If the channel capacity is b bits/sec, the frame size l bits, and the round-trip propagation time R sec, the time required to transmit a single frame is l/b sec. After the last bit of a data frame has been sent, there is a delay of R/2 before that bit arrives at the receiver and another delay of at least R/2 for the acknowledgement to come back, for a total delay of R. In stop-and-wait the line is busy for l/band idle for R, giving

If $l < bR$, the efficiency will be less than 50 percent. Since there is always a nonzero delay for the

acknowledgement to propagate back, pipelining can, in principle, be used to keep the line busy during this interval, but if the interval is small, the additional complexity is not worth the trouble.

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

In Fig.10.1 (a) we see go back n for the case in which the receiver's window is large. Frames 0 and 1 are correctly received and acknowledged. Frame 2, however, is damaged or lost. The sender, unaware of this problem, continues to send frames until the timer for frame 2 expires. Then it backs up to frame 2 and starts all over with it, sending 2, 3, 4, etc. all over again. The other general strategy for handling errors when frames are pipelined is called selective repeat. When it is used, a bad frame that is received is discarded, but good frames received after it are buffered. When the sender times out, only the oldest unacknowledged frame is retransmitted. If that frame arrives correctly, the receiver can deliver to the network layer, in sequence, all the frames it has buffered. Selective repeat is often combined with having the receiver send a negative acknowledgement (NAK) when it detects an error, for example, when it receives a checksum error or a frame out of sequence. NAKs stimulate retransmission before the corresponding timer expires and thus improve performance. In Fig.10.1 (b), frames 0 and 1 are again correctly received and acknowledged and frame 2 is lost. When frame 3 arrives at the receiver, the data link layer there notices that is has missed a frame, so it sends back a NAK for 2 but buffers 3. When frames 4 and 5 arrive, they, too, are buffered by the data link layer instead of being passed to the network layer. Eventually, the NAK 2 gets back to the sender, which

immediately resends frame 2. When that arrives, the data link layer now has 2, 3, 4, and 5 and can pass all of them to the network layer in the correct order. It can also acknowledge all frames up to and including 5, as shown in the figure. If the NAK should get lost, eventually the sender will time out for frame 2 and send it (and only it) of its own accord, but that may be a quite a while later. In effect, the NAK speeds up the retransmission of one specific frame.

Selective repeat corresponds to a receiver window larger than 1. Any frame within the window may be accepted and buffered until all the preceding ones have been passed to the network layer. This approach can require large amounts of data link layer memory if the window is large. These two alternative approaches are trade-offs between bandwidth and data link layer buffer space. Depending on which resource is scarcer, one or the other can be used. Figure 10.2 shows a pipelining protocol in which the receiving data link layer only accepts frames in order; frames following an error are discarded. In this protocol, for the first time we have dropped the assumption that the network layer always has an infinite supply of packets to send. When the network layer has a packet it wants to send, it can cause a network_layer_ready event to happen. However, to enforce the flow control rule of no more than MAX_SEQ unacknowledged frames outstanding at any time, the data link layer must be able to keep the network layer from bothering it with more work. The library procedures enable_network_layer and disable_network_layer do this job.

A Protocol Using Selective Repeat

This protocol works well if errors are rare, but if the line is poor, it wastes a lot of bandwidth on retransmitted frames. An alternative strategy for handling errors is to allow the receiver to accept and buffer the frames following a damaged or lost one. Such a protocol does not discard frames merely because an earlier frame was damaged or lost. In this protocol, both sender and receiver maintain a window of acceptable sequence numbers. The sender's window size starts out at 0 and grows to some predefined maximum, MAX_SEQ. The receiver's window, in contrast, is always fixed in size and equal to MAX_SEQ. The receiver has a buffer reserved for each sequence number within its fixed window. Associated with each buffer is a bit (arrived) telling whether the buffer is full or empty. Whenever a frame arrives, its sequence number is checked by the function between to see if it falls within the window. If so and if it has not already been received, it is accepted and stored. This action is taken without regard to whether or not it contains the next

packet expected by the network layer. Of course, it must be kept within the data link layer and not passed to the network layer until all the lower-numbered frames have already been delivered to the network layer in the correct order.

5.1 Random Access:

5.1.1 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. Their work has been extended by many researchers since then (Abramson, 1985).

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. There are two versions of ALOHA: 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:

The basic idea of an ALOHA system is simple: let users transmit whenever they have data to be sent. There will be collisions, of course, and the colliding frames will be damaged. However, due to the feedback property of broadcasting, a sender can always find out whether its frame was destroyed by listening to the channel, the same way other users do. With a LAN, the feedback is immediate; with a satellite, there is a delay of 270 msec before the sender knows if the transmission was successful. If listening while transmitting is not possible for some reason, acknowledgements are needed. If the frame was destroyed, the sender just waits a random amount of time and sends it again. The waiting time must be random or the same frames will collide over and over, in lockstep. Systems in which multiple users share a common channel in a way that can lead to conflicts are widely known as contention systems.

A sketch of frame generation in an ALOHA system is given in Fig.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.

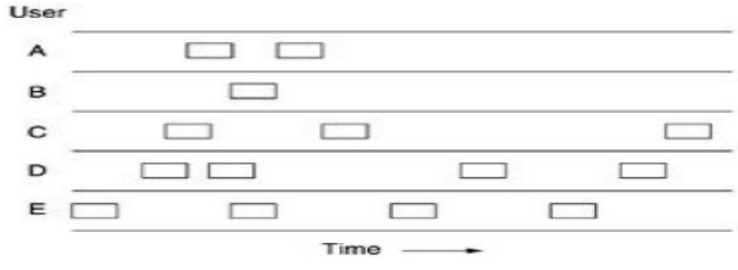


Fig.1 In pure ALOHA, frames are transmitted at completely arbitrary times.

Whenever two frames try to occupy the channel at the same time, there will be a collision and both will be garbled. If the first bit of a new frame overlaps with just the last bit of a frame almost finished, both frames will be totally destroyed and both will have to be retransmitted later. The checksum cannot (and should not) distinguish between a total loss and a near miss.

Let the "frame time" denote the amount of time needed to transmit the standard, fixed length frame (i.e., the frame length divided by the bit rate). At this point we assume that the infinite population of users generates new frames according to a Poisson distribution with mean N frames per frame time. (The infinite-population assumption is needed to ensure that N does no decrease as users become blocked.) If $N > 1$, the user community is generating frames at a higher rate than the channel can handle, and nearly every frame will suffer a collision. For reasonable throughput we would expect $0 < N < 1$. In addition to the new frames, the stations also generate retransmissions of frames that previously suffered collisions. Let us further assume that the probability of k transmission attempts per frame time, old and new combined, is also Poisson, with mean G per frame time. Clearly, $G \geq N$. At low load (i.e., N 0), there will be few collisions, hence few retransmissions, so G N. At high load there will be many collisions, so $G > N$. Under all loads, the throughput, S, is just the offered load, G, times the probability, P0, of a transmission succeeding—that is, $S = GP0$, where P0 is the probability that a frame does not suffer a collision.

A frame will not suffer a collision if no other frames are sent within one frame time of its start, as shown in Fig.2.

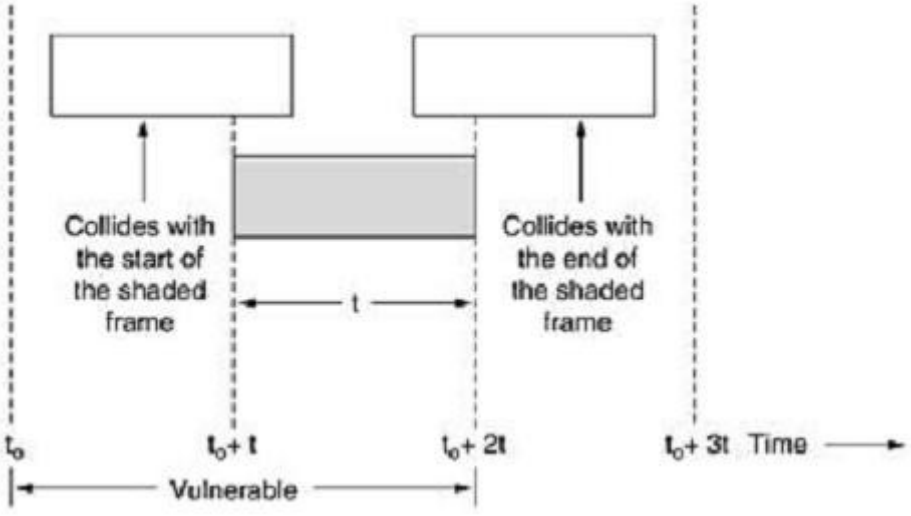


Fig.2. Vulnerable period for the shaded frame

Under what conditions will the shaded frame arrive undamaged? Let t be the time required to send a frame. If any other user has generated a frame between time t0 and t0+tr, the end of that frame will collide with the beginning of the shaded one. In fact, the shaded frame's fate was already sealed even before the first bit was sent, but since in pure ALOHA a station does not listen to the channel before transmitting, it has no way of knowing that another frame was already underway. Similarly, any other frame started between t0+t and t0+2t will bump into the end of the shaded frame.

The probability that k frames are generated during a given frame time is given by the Poisson distribution:

Equation

$$\Pr[k] = \frac{G^k e^{-G}}{k!}$$

so the probability of zero frames is just e-G. In an interval two frame times long, the mean number of frames generated is 2G. The probability of no other traffic being initiated during the entire vulnerable period is thus given by P0 = e -2G. Using S = GP0, we get

$$S = Ge^{-2G}$$

The relation between the offered traffic and the throughput is shown in Fig. 4-3. The maximum throughput occurs at G = 0.5, with S = 1/2e, which is about 0.184. In other words, the best we can hope for is a channel utilization of 18 per cent. This result is not very encouraging, but with everyone transmitting at will, we could hardly have expected a 100 per cent success rate.

Slotted ALOHA:

In 1972, Roberts published a method for doubling the capacity of an ALOHA system (Robert, 1972). His proposal was to divide time into discrete intervals, each interval corresponding to one frame. This approach requires the users to agree on slot boundaries. One way to achieve synchronization would be to have one special station emit a pip at the start of each interval, like a clock.

In Roberts' method, which has come to be known as slotted ALOHA, in contrast to Abramson's pure ALOHA, a computer is not permitted to send whenever a carriage return is typed. Instead, it is required to wait for the beginning of the next slot. Thus, the continuous pure ALOHA is turned

into a discrete one. Since the vulnerable period is now halved, the probability of no other traffic during the same slot as our test frame is e-G which leads to

\Equation

$$S = Ge^{-G}$$

As you can see from Fig.3, slotted ALOHA peaks at G = 1, with a throughput of S=1/e or about 0.368, twice that of pure ALOHA. If the system is operating at G = 1, the probability of an empty slot is 0.368. The best we can hope for using slotted ALOHA is 37 percent of the slots empty, 37 percent successes, and 26 percent collisions. Operating at higher values of G reduces the number of empties but increases the number of collisions exponentially.

To see how this rapid growth of collisions with G comes about, consider the transmission of a test frame. The probability that it will avoid a collision is e-G, the probability that all the other users are silent in that slot. The probability of a collision is then just 1 – e-G. The probability of a transmission requiring exactly k attempts, (i.e., k – 1 collisions followed by one success) is

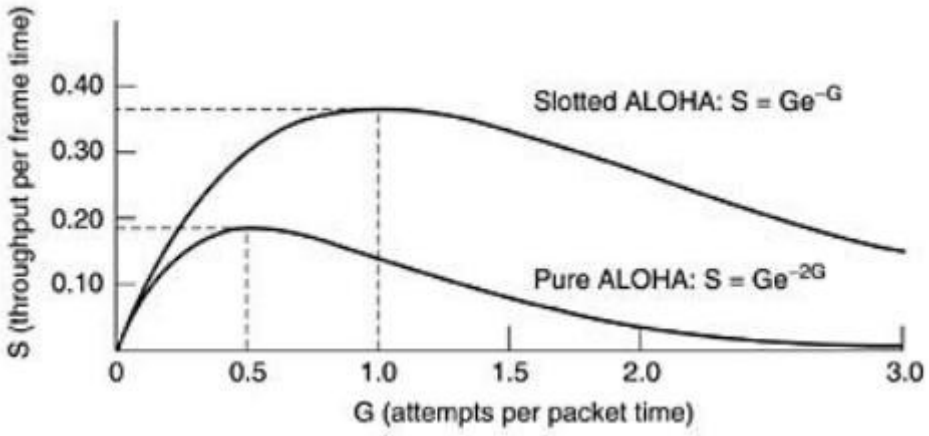


Fig.3 Throughput versus offered traffic for ALOHA systems.

$$P_k = e^{-G}(1 - e^{-G})^{k-1}$$

The expected number of transmissions, E, per carriage return typed is then

$$E = \sum_{k=1}^{\infty} kP_k = \sum_{k=1}^{\infty} ke^{-G}(1 - e^{-G})^{k-1} = e^{G}$$

As a result of the exponential dependence of E upon G, small increases in the channel load can drastically reduce its performance.

5.1.2 CSMA

Carrier Sense Multiple Access Protocols:

With slotted ALOHA the best channel utilization that can be achieved is 1/e. This is hardly surprising, since with stations transmitting at will, without paying attention to what the other stations are doing, there are bound to be many collisions. In local area networks, however, it is possible for stations to detect what other stations are doing, and adapt their behaviour accordingly. These networks can achieve a much better utilization than 1/e. In this section we will discuss some protocols for improving performance. Protocols in which stations listen for a carrier (i.e., a transmission) and act accordingly are called carrier sense protocols. A number of them have been proposed. Kleinrock and Tobagi (1975) have analysed several such protocols in detail. Below we will mention several versions of the carrier sense protocols.

1.1-persistent CSMA:

The first carrier sense protocol that we will study here is called **1-persistent CSMA** (Carrier Sense Multiple Access). When a station has data to send, it first listens to the channel to see if anyone else is transmitting at that moment. If the channel is busy, the station waits until it becomes idle. When the station detects an idle channel, it transmits a frame. If a collision occurs, the station waits a random amount of time and starts all over again. The protocol is called 1-persistent because the station transmits with a probability of 1 when it finds the channel idle.

The propagation delay has an important effect on the performance of the protocol. There is a small chance that just after a station begins sending, another station will become ready to send and sense the channel. If the first station's signal has not yet reached the second one, the latter will sense an idle channel and will also begin sending, resulting in a collision. The longer the propagation delay, the more important this effect becomes, and the worse the performance of the protocol. Even if the propagation delay is zero, there will still be collisions. If two stations become ready in the middle of a third station's transmission, both will wait politely until the transmission ends and then both will begin transmitting exactly simultaneously, resulting in a collision. If they were not so impatient, there would be fewer collisions. Even so, this protocol is far better than pure ALOHA because both stations have the decency to desist from interfering with the third station's frame. Intuitively, this approach will lead to a higher performance than pure ALOHA. Exactly the same holds for slotted ALOHA.

2. Non-persistent CSMA:

A second carrier sense protocol is **nonpersistent CSMA**. In this protocol, a conscious attempt is made to be less greedy than in the previous one. Before sending, a station senses the channel. If no one else is sending, the station begins doing so itself. However, if the channel is already in use, the station does not continually sense it for the purpose of seizing it immediately upon detecting the end of the previous transmission. Instead, it waits a random period of time and then repeats the algorithm. Consequently, this algorithm leads to better channel utilization but longer delays than 1-persistent CSMA.

3. P-persistent CSMA:

The last protocol is **p-persistent CSMA**. It applies to slotted channels and works as follows. When a station becomes ready to send, it senses the channel. If it is idle, it transmits with a probability p. With a probability q = 1 - p, it defers until the next slot. If that slot is also idle, It either transmits or defers again, with probabilities p and q. This process is repeated until either the frame has been transmitted or another station has begun transmitting. In the latter case, the unlucky station acts as if there had been a collision (i.e., it waits a random time and starts again). If the station initially senses the channel busy, it waits until the next slot and applies the above algorithm. Figure 4 shows the computed throughput versus offered traffic for all three protocols, as well as for pure and slotted ALOHA.

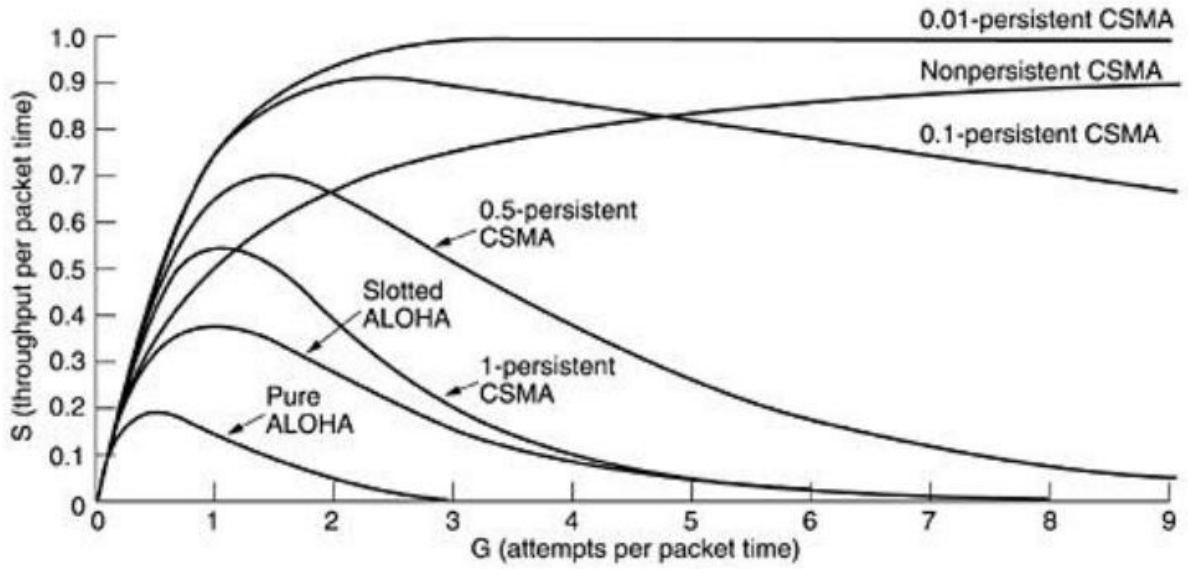


Fig.4 Comparison of the channel utilization versus load for various random access protocols

5.1.3 CSMA with Collision Detection:

Persistent and nonpersistent CSMA protocols are clearly an improvement over ALOHA because they ensure that no station begins to transmit when it senses the channel busy. Another improvement is for stations to abort their transmissions as soon as they detect a collision. In other words, if two stations sense the channel to be idle and begin transmitting simultaneously, they will both detect the collision almost immediately. Rather than finish transmitting their frames, which are irretrievably garbled anyway, they should abruptly stop transmitting as soon as the collision is detected. Quickly terminating damaged frames saves time and bandwidth.

This protocol, known as CSMA/CD (CSMA with Collision Detection) is widely used on LANs in the MAC sublayer. In particular, it is the basis of the popular Ethernet LAN, so it is worth devoting some time to looking at it in detail. CSMA/CD, as well as many other LAN protocols, uses the conceptual model of Fig.5. At the point marked t0, a station has finished transmitting its frame. Any other station having a frame to send may now attempt to do so. If two or more stations decide to transmit simultaneously, there will be a collision. Collisions can be detected by looking at the power or pulse width of the received signal and comparing it to the transmitted signal.

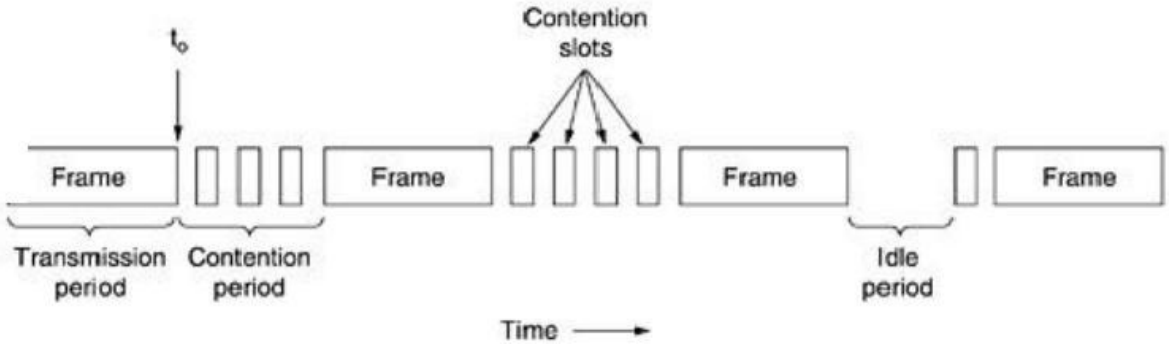


Fig.5. CSMA/CD can be in one of three states: contention, transmission, or idle

After a station detects a collision, it aborts its transmission, waits a random period of time, and then tries again, assuming that no other station has started transmitting in the meantime. Therefore, our model for CSMA/CD will consist of alternating contention and transmission periods, with idle periods occurring when all stations are quiet (e.g., for lack of work).

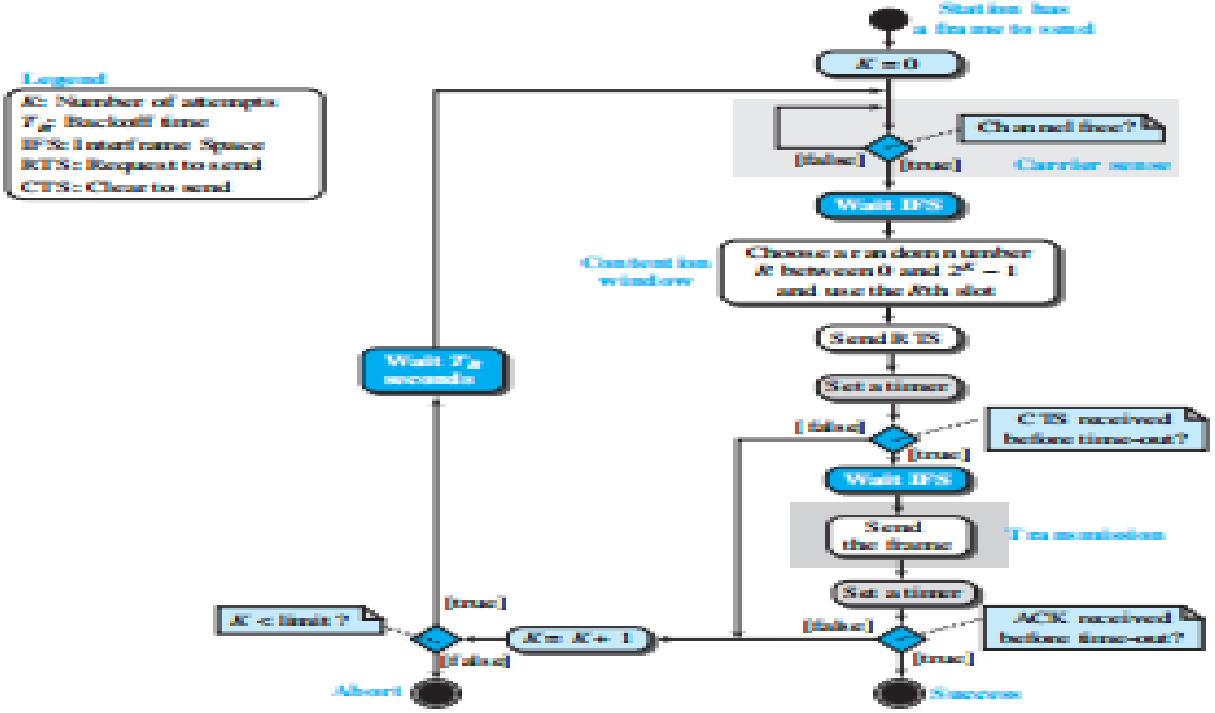
Now let us look closely at the details of the contention algorithm. Suppose that two stations both begin transmitting at exactly time t0. How long will it take them to realize that there has been a collision? The answer to this question is vital to determining the length of the contention period and hence what the delay and throughput will be. The minimum time to detect the collision is then just the time it takes the signal to propagate from one station to the other.

Based on this reasoning, you might think that a station not hearing a collision for a time equal to the full cable propagation time after starting its transmission could be sure it had seized the cable. By "seized," we mean that all other stations knew it was transmitting and would not interfere. This conclusion is wrong. Consider the following worst-case scenario. Let the time for a signal to propagate between the two farthest stations be . At t0, one station begins transmitting. At , an instant before the signal arrives at the most distant station, that station also begins transmitting. Of course, it detects the collision almost instantly and stops, but the little noise burst caused by the collision does not get back to the original station until time . In other words, in the worst case a station cannot be sure that it has seized the channel until it has transmitted for without hearing a collision. For this reason we will model the contention interval as a slotted ALOHA system with slot width . On a 1-km long coaxial cable, . For simplicity we will assume that each slot contains just 1 bit. Once the channel has been seized, a station can transmit at any rate it wants to, of course, not just at 1 bit per sec.

CSMA/CA

Carrier sense multiple access with collision avoidance (CSMA/CA) was invented for wireless networks. Collisions are avoided through the use of CSMA/CA's three strategies: the interframe space, the contention window, and acknowledgments, as shown in Figure 12.15. We discuss RTS and CTS frames later.

Figure 12.15 *Flow diagram of CSMA/CA*



6.1 IEEE STANDARDS

In 1985, the Computer Society of the IEEE started a project, called Project 802, to set standards to enable intercommunication among equipment from a variety of manufacturers. Project 802 does not seek to replace any part of the OSI or the Internet model. Instead, it is a way of specifying functions of the physical layer and the data link layer of major LAN protocols.

The standard was adopted by the American National Standards Institute (ANSI). In 1987, the International Organization for Standardization (ISO) also approved it as an international standard under the designation ISO 8802. The relationship of the 802 Standard to the traditional OSI model is shown in Figure 1.

The IEEE has subdivided the data link layer into two sublayers:

logical link control (LLC) and media access control (MAC).

IEEE has also created several physical layer standards for different LAN protocols.

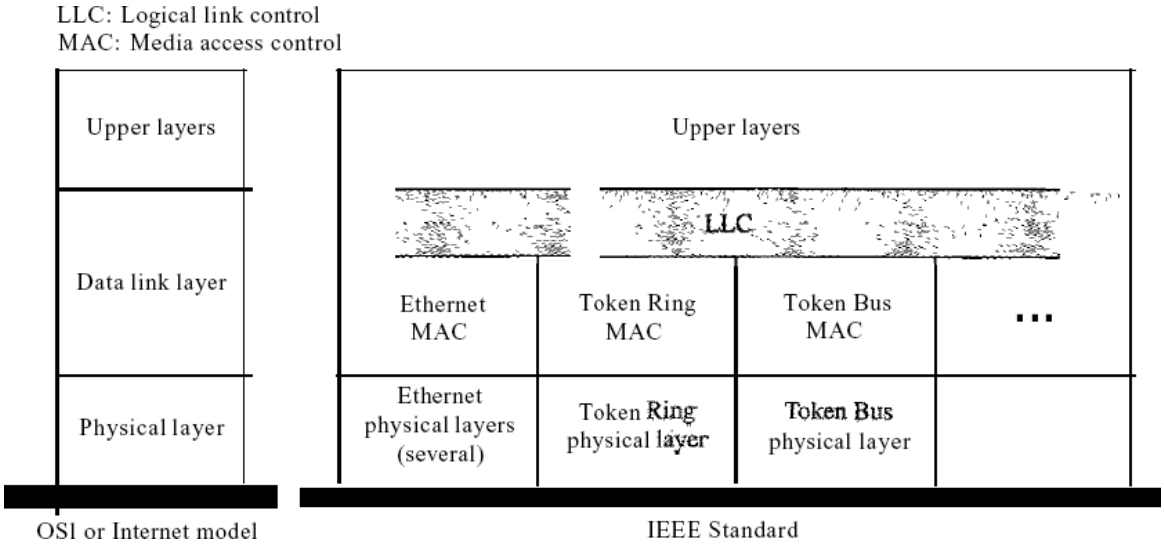


Figure 1 *IEEE standardfor LANs*

6.1.1 Data Link Layer

As we mentioned before, the data link layer in the IEEE standard is divided into two sublayers: LLC and MAC.

Logical Link Control (LLC)

We said that data link control handles framing, flow control, and error control. In IEEE Project 802, flow control, error control,and part of the framing duties are collected into one sublayer called the logical link control. Framing is handled in both the LLC sublayer and the MAC sublayer.

The LLC provides one single data link control protocol for all IEEE LANs. In this way, the LLC is different from the media access control sublayer, which provides different protocols for different LANs. A single LLC protocol can provide interconnectivity between different LANs because it makes the MAC sublayer transparent. Figure 1 shows one single LLC protocol serving several MAC protocols. *Media Access Control (MAC)*

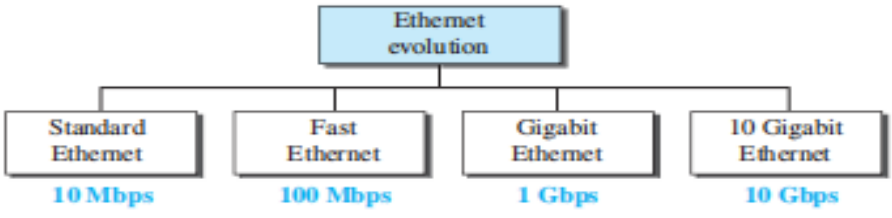
IEEE Project 802 has created a sublayer called media access control that defines the specific access method for each LAN. For example, it defines *CSMA/CD* as the media access method for Ethernet LANs and the tokenpassing method for Token Ring and Token Bus LANs. As we discussed in the previous section, part of the framing function is also handled by the MAC layer. In contrast to the LLC sublayer, the MAC sublayer contains a number of distinct modules; each defines the access method and the framing format specific to the corresponding LAN protocol.

6.1.2 Physical Layer

The physical layer is dependent on the implementation and type of physical media used. IEEE defines detailed specifications for each LAN implementation. For example, although there is only one MAC sublayer for Standard Ethernet, there is a different physical layer specifications for each Ethernet implementations.

6.2 Standard Ethernet

Figure 13.2 *Ethernet evolution through four generations*



MAC Sublayer

In Standard Ethernet, the MAC sublayer governs the operation of the access method. It also

frames data received from the upper layer and passes them to the physical layer.

Frame Format

The Ethernet frame contains seven fields: preamble, SFD, DA, SA, length or type of protocol data unit (PDU), upper-layer data, and the CRe. Ethernet does not provide any mechanism for acknowledging received frames, making it what is known as an unreliable medium.

Acknowledgments must be implemented at the higher layers. The format ofthe MAC frame is shown in Figure 4.

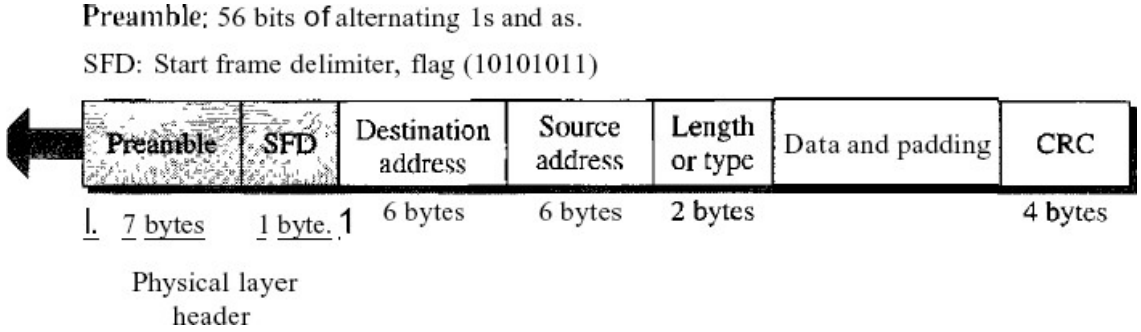


Figure 4 *802.3 MACframe*

- D Preamble. The first field of the 802.3 frame contains 7 bytes (56 bits) of alternating Os and Is that alerts the receiving system to the coming frame and enables it to synchronize its input timing. The pattern provides only an alert and a timing pulse. The 56-bit pattern allows the stations to miss some bits at the beginning of the frame. The preamble is actually added at the physical layer and is not (formally) part of the frame.
- D Start frame delimiter (SFD). The second field (l byte: 10101011) signals the beginning of the frame. The SFD warns the station or stations that this is the last chance for synchronization. The last 2 bits is 11 and alerts the receiver that the next field is the destination address.
- Destination address (DA). The DA field is 6 bytes and contains the physical address of the destination station or stations to receive the packet.
- Source address (SA). The SA field is also 6 bytes and contains the physical address of the sender of the packet. We will discuss addressing shortly.
- Length or type. This field is defined as a type field or length field. The original Ethernet used this field as the type field to define the upper-layer protocol using the MAC frame. The IEEE standard used it as the length field to define the number of bytes in the data field. Both uses are common today.
- Data. This field carries data encapsulated from the upper-layer protocols. It is a minimum of 46 and a maximum of 1500 bytes.
- CRC. The last field contains error detection information, in this case a CRC-32

Frame Length

Ethernet has imposed restrictions on both the minimum and maximum lengths of a frame, as shown in Figure 5.

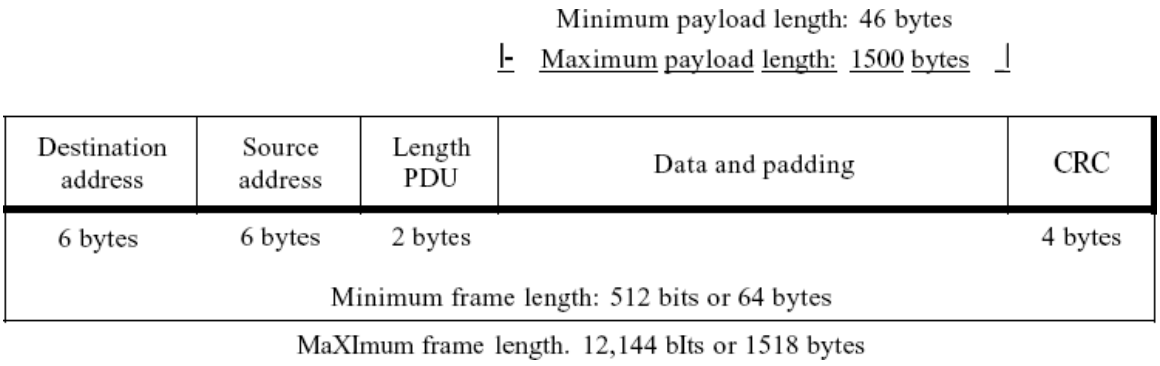


Figure 5 *Minimum and maximum lengths*

The minimum length restriction is required for the correct operation of *CSMAlCD*. An Ethernet frame needs to have a minimum length of 512 bits or 64 bytes. Part of this length is the header and the trailer. If we count 18 bytes of header and trailer (6 bytes of source address, 6 bytes of destination address, 2 bytes of length or type, and 4 bytes of CRC), then the minimum length of data from the upper layer is 64 - 18 = 46 bytes. If the upper-layer packet is less than 46 bytes, padding is added to make up the difference.

The standard defines the maximum length of a frame (without preamble and SFD field) as 1518 bytes. If we subtract the 18 bytes of header and trailer, the maximum length of the payload is 1500 bytes. The maximum length restriction has two historical reasons. First, memory was very expensive when Ethernet was designed: a maximum length restriction helped to reduce the size of the buffer. Second, the maximum length restriction prevents one station from monopolizing the shared medium, blocking other stations that have data to send.

Frame length:

Minimum: 64 bytes (512 bits) Maximum: 1518 bytes (12,144 bits)

Addressing

Each station on an Ethernet network (such as a PC, workstation, or printer) has its own network interface card (NIC). The NIC fits inside the station and provides the station with a 6-byte

physical address. As shown in Figure 6, the Ethernet address is 6 bytes (48 bits), nonnally written in hexadecimal notation, with a colon between the bytes.

06:01 :02:01:2C:4B

6 bytes =12 hex digits =48 bits

Figure 6 *Example ofan Ethernet address in hexadecimal notation*

Unicast, Multicast, and Broadcast Addresses A source address is always a unicast address-the frame comes from only one station. The destination address, however, can be unicast, multicast, or broadcast. Figure 7 shows how to distinguish a unicast address from a multicast address. If the least significant bit of the first byte in a destination address is 0, the address is unicast; otherwise, it is multicast.

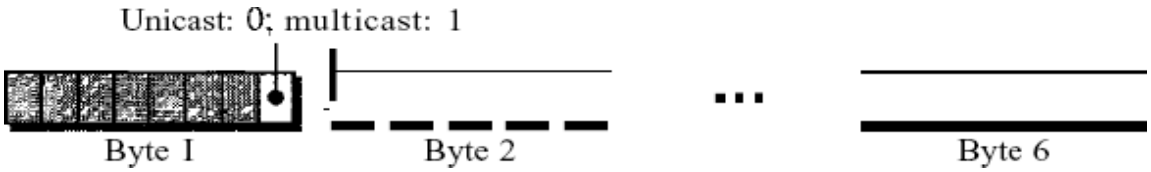


Figure 7 *Unicast and multicast addresses*

A unicast destination address defines only one recipient; the relationship between the sender and the receiver is one-to-one. A multicast destination address defines a group of addresses; the relationship between the sender and the receivers is one-to-many. The broadcast address is a special case of the multicast address; the recipients are all the stations on the LAN. A broadcast destination address is forty-eight.

6.2.1 Physical Layer

The Standard Ethernet defines several physical layer implementations; four of the most common, are shown in Figure 8.

Encoding and Decoding

All standard implementations use digital signaling (baseband) at 10 Mbps. At the sender, data are converted to a digital signal using the Manchester scheme; at the receiver, the received signal is interpreted as Manchester and decoded into data. Manchester encoding is self-synchronous, providing a transition at each bit interval. Figure 9 shows the encoding scheme for Standard Ethernet.

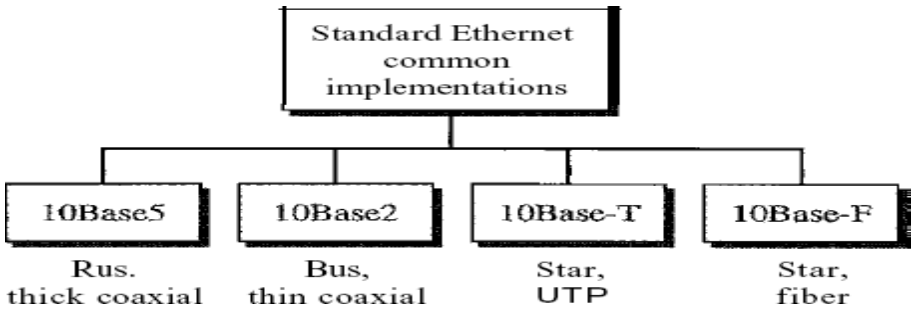


Figure 8 *Categories of Standard Ethernet*

Fast Ethernet

Fast Ethernet was designed to compete with LAN protocols such as FDDI or Fiber Channel (or Fibre Channel, as it is sometimes spelled). IEEE created Fast Ethernet under the name 802.3u. Fast Ethernet is backward-compatible with Standard Ethernet, but it can transmit data 10 times faster at a rate of 100 Mbps.

The goals of Fast Ethernet can be summarized as follows:

1. Upgrade the data rate to 100 Mbps.
2. Make it compatible with Standard Ethernet.
3. Keep the same 48-bit address.
4. Keep the same frame format.
5. Keep the same minimum and maximum frame lengths.

13.4 GIGABIT ETHERNET

The need for an even higher data rate resulted in the design of the Gigabit Ethernet Protocol (1000 Mbps). The IEEE committee calls it the Standard 802.3z. The goals of the Gigabit Ethernet were to upgrade the data rate to 1 Gbps, but keep the address length, the frame format, and the maximum and minimum frame length the same. The goals of the Gigabit Ethernet design can be summarized as follows:

1. Upgrade the data rate to 1 Gbps.
2. Make it compatible with Standard or Fast Ethernet.
3. Use the same 48-bit address.
4. Use the same frame format.
5. Keep the same minimum and maximum frame lengths.
6. Support autonegotiation as defined in Fast Ethernet

13.4.1 MAC Sublayer

A main consideration in the evolution of Ethernet was to keep the MAC sublayer untouched. However, to achieve a data rate of 1 Gbps, this was no longer possible. Giga
bit Ethernet has two distinctive approaches for medium access: half-duplex and full duplex. Almost all implementations of Gigabit Ethernet follow the full-duplex approach,
so we mostly ignore the half-duplex mode.

Full-Duplex Mode

In full-duplex mode, there is a central switch connected to all computers or other switches. In this mode, for each input port, each switch has buffers in which data are stored until they are transmitted. Since the switch uses the destination address of the frame and sends a frame out of the port connected to that particular destination, there is
no collision. This means that CSMA/CD is not used. Lack of collision implies that the maximum length of the cable is determined by the signal attenuation in the cable, not by
the collision detection process.

Half-Duplex Mode

Gigabit Ethernet can also be used in half-duplex mode, although it is rare. In this case, a switch can be replaced by a hub, which acts as the common cable in which a collision might occur. The half-duplex approach uses CSMA/CD. However, as we saw before,the maximum length of the network in this approach is totally dependent on the minimum frame size. Three methods have been defined: traditional, carrier extension, and frame bursting.

Traditional

In the traditional approach, we keep the minimum length of the frame as in traditional Ethernet (512 bits). However, because the length of a bit is 1/100 shorter in Gigabit Ethernet than in 10-Mbps Ethernet, the slot time for Gigabit Ethernet is 512 bits × 1/ 1000 μs, which is equal to 0.512 μs. The reduced slot time means that collision is detected
100 times earlier. This means that the maximum length of the network is 25 m. This length may be suitable if all the stations are in one room, but it may not even be long enough to connect the computers in one single office.

Carrier Extension

To allow for a longer network, we increase the minimum frame length. The **carrier extension** approach defines the minimum length of a frame as 512 bytes (4096 bits). This means that the minimum length is 8 times longer. This method forces a station to add extension bits (padding) to any frame that is less than 4096 bits. In this way, the maximum length of the network can be increased 8 times to a length of 200 m. This allows a length of 100 m from the hub to the station.

Frame Bursting

Carrier extension is very inefficient if we have a series of short frames to send; each frame carries redundant data. To improve efficiency, **frame bursting** was proposed. Instead of adding an extension to each frame, multiple frames are sent. However, to make these multiple frames look like one frame, padding is added between the frames (the same as that used for the carrier extension method) so that the channel is not idle. In other words, the method deceives other stations into thinking that a very large frame has been transmitted

13.4.2 Physical Layer

The physical layer in Gigabit Ethernet is more complicated than that in Standard or Fast Ethernet. We briefly discuss some features of this layer.

Topology

Gigabit Ethernet is designed to connect two or more stations. If there are only two stations, they can be connected point-to-point. Three or more stations need to be connected in a star topology with a hub or a switch at the center. Another possible configuration is to connect several star topologies or let one star topology be part of another.

Implementation

Gigabit Ethernet can be categorized as either a two-wire or a four-wire implementation.

The two-wire implementations use fiber-optic cable (**1000Base-SX,** short-wave, or **1000Base-LX,** long-wave), or STP (**1000Base-CX**). The four-wire version uses category 5 twisted-pair cable (**1000Base-T**). In other words, we have four implementations.

1000Base-T was designed in response to those users who had already installed this wiring for other purposes such as Fast Ethernet or telephone services.

Encoding

Figure 13.17 shows the encoding/decoding schemes for the four implementations.

Gigabit Ethernet cannot use the Manchester encoding scheme because it involves a very high bandwidth (2 GBaud). The two-wire implementations use an NRZ scheme, but NRZ does not self-synchronize properly. To synchronize bits, particularly at this high data rate, 8B/10B block encoding, discussed in Chapter 4, is used.

This block encoding prevents long sequences of 0s or 1s in the stream, but the result ing stream is 1.25 Gbps. Note that in this implementation, one wire (fiber or STP) is used for sending and one for receiving.

In the four-wire implementation it is not possible to have 2 wires for input and 2 for output, because each wire would need to carry 500 Mbps, which exceeds the capacity for category 5 UTP. As a solution, 4D-PAM5 encoding, as discussed in Chapter 4, is used to reduce the bandwidth. Thus, all four wires are involved in both input and output; each wire carries 250 Mbps, which is in the range for category 5 UTP cable.

13.5 10 GIGABIT ETHERNET

In recent years, there has been another look into the Ethernet for use in metropolitan areas. The idea is to extend the technology, the data rate, and the coverage distance so that the Ethernet can be used as LAN and MAN (metropolitan area network). The IEEE

committee created 10 Gigabit Ethernet and called it Standard 802.3ae. The goals of the 10 Gigabit Ethernet design can be summarized as upgrading the data rate to 10 Gbps, keeping the same frame size and format, and allowing the interconnection of LANs, MANs, and WAN possible. This data rate is possible only with fiber-optic technology at this time. The standard defines two types of physical layers: LAN PHY and WAN PHY. The first is designed to support existing LANs; the second actually defines a WAN with links connected through SONET OC-192.

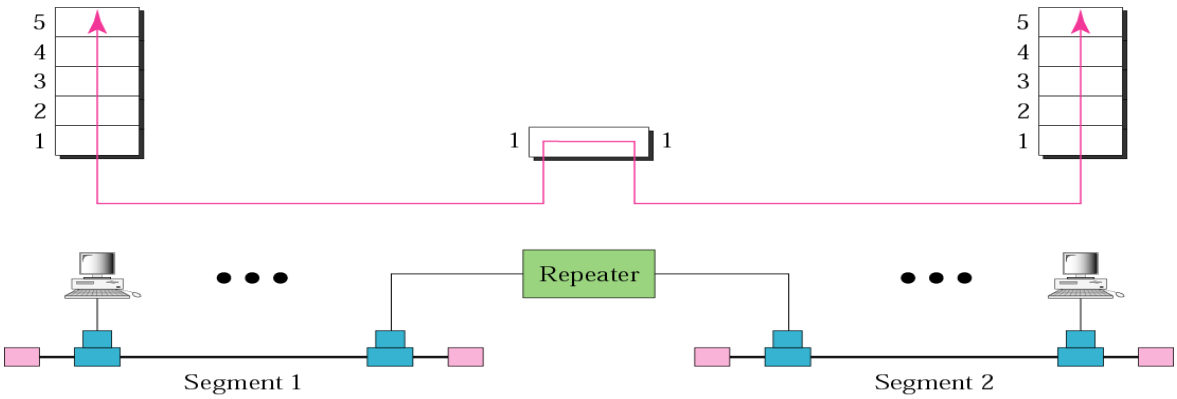
Connecting Devices:

Devices that are used to connect other devices are called connecting devices. That is, connecting device is a device that connects.

There are 4 types of connecting devices. They are as follows,

i. Repeater:

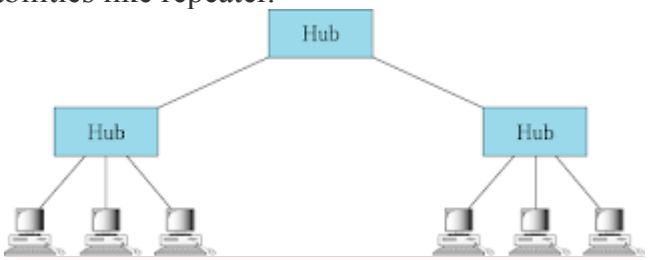
A repeater is a device that receives signals and before it become too weak or corrupted the repeater regenerates the original bit pattern and sends the data. It is like a refreshing station for data while travelling. Some properties are as follows,



1. It operates only in the physical layer.
2. It connects two segments of the same LAN.
3. It forwards every bits it receives.
4. It regenerate bits, does not amplify them.

ii. Hub:

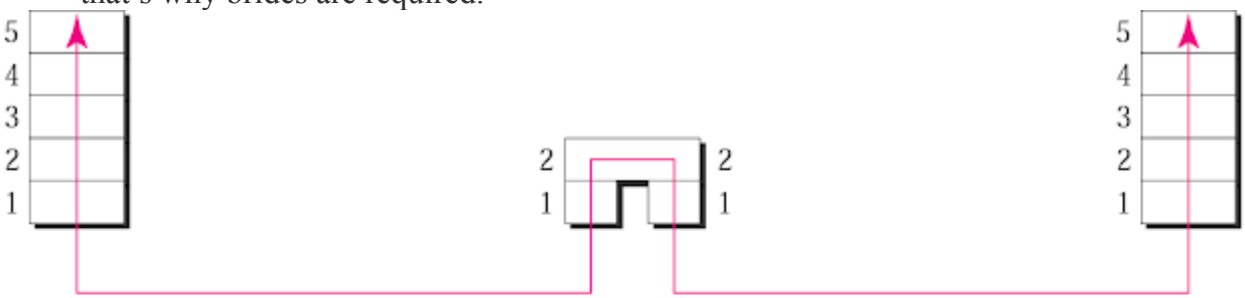
A hub is a multi-port repeater. That is a hub can connect more than two segments of a LAN where a repeater can connect only two segments. It also does not have any filtering capabilities like repeater.



iii. Bridge:

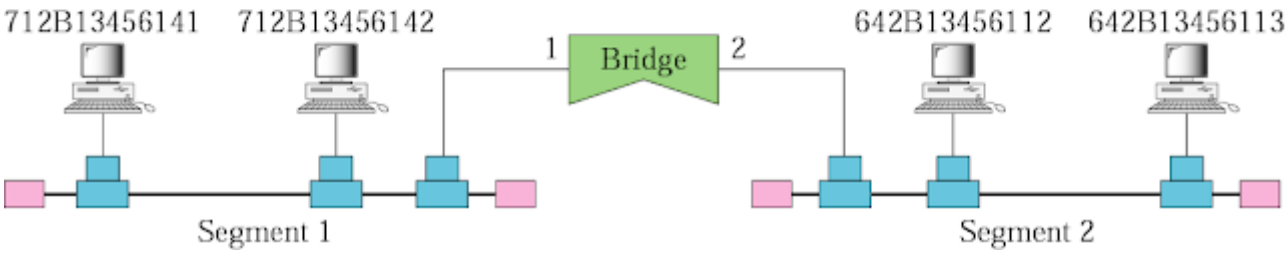
A bridge operates in both physical layer and data link layer. As a physical layer device, it can regenerate bits and as a data link layer device it can check the physical addresses contained in the frame. Bits can be carried to any distance by

using properly spaced repeaters but the collision domain will not allow this and that's why brides are required.



Address	Interface
712B13456141	1
712B13456142	1
642B13456112	2
642B13456113	2

Bridge table



Bridge has filtering capability. It has table that maps addresses to port. A bridge can check the table to decide if a frame has to be forwarded or dropped. Bridges also work in a single LAN. There are some bridges called transparent bridge which can generate its address table automatically by learning the frame movement in the network.

IV switch

Transparent Switches

A **transparent switch** is a switch in which the stations are completely unaware of the switch's existence. If a switch is added or deleted from the system, reconfiguration of the stations is unnecessary. According to the IEEE 802.1d specification, a system equipped with transparent switches must meet three criteria:

Frames must be forwarded from one station to another.

The forwarding table is automatically made by learning frame movements in the network.

Loops in the system must be prevented.

Forwarding

A transparent switch must correctly forward the frames, as discussed in the previous section.

Learning

The earliest switches had switching tables that were static. The system administrator would manually enter each table entry during switch setup. Although the process was simple, it was not practical. If a station was added or deleted, the table had to be modified

manually. The same was true if a station's MAC address changed, which is not a rare event. For example, putting in a new network card means a new MAC address.
A better solution to the static table is a dynamic table that maps addresses to ports (interfaces) automatically. To make a table dynamic, we need a switch that gradually learns from the frames' movements. To do this, the switch inspects both the destination
and the source addresses in each frame that passes through the switch. The destination address is used for the forwarding decision (table lookup); the source address is used for adding entries to the table and for updating purposes.

1. When station A sends a frame to station D, the switch does not have an entry for either D or A. The frame goes out from all three ports; the frame floods the network. However, by looking at the source address, the switch learns that station A must be connected to port 1. This means that frames destined for A, in the future, must be sent out through port 1. The switch adds this entry to its table. The table has its first entry now.

2. When station D sends a frame to station B, the switch has no entry for B, so it floods the network again. However, it adds one more entry to the table related to station D.

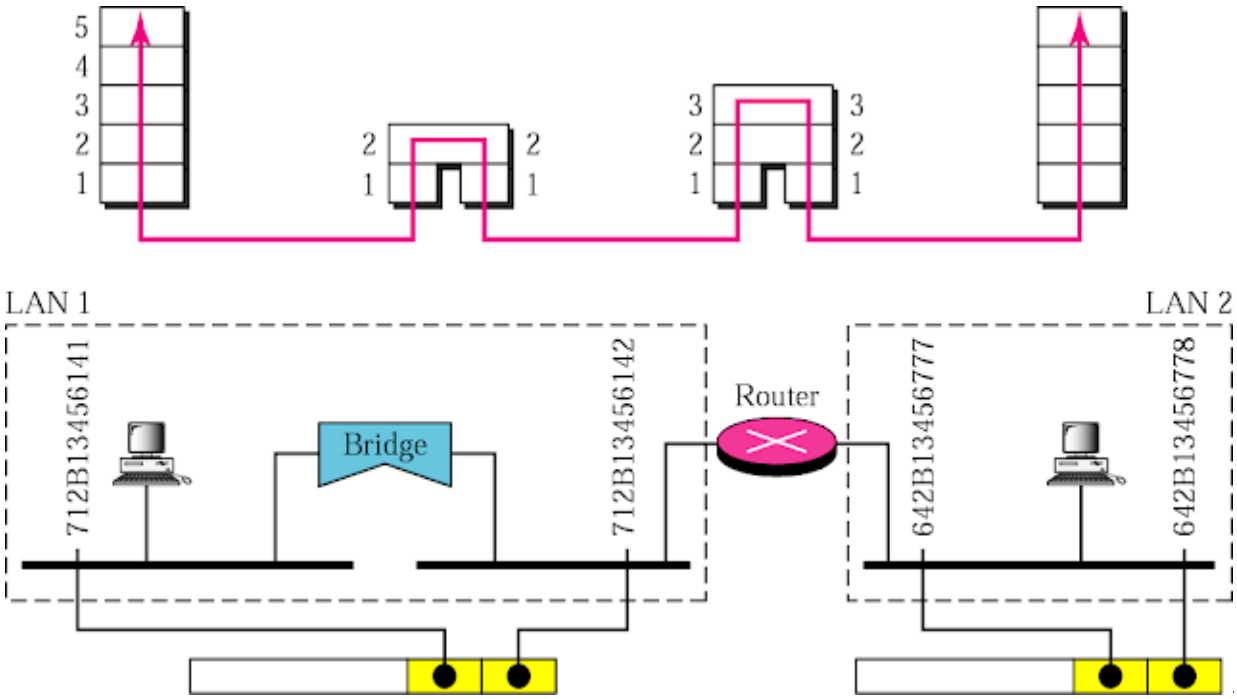
3. The learning process continues until the table has information about every port. However, note that the learning process may take a long time. For example, if a station does not send out a frame (a rare situation), the station will never have an entry in the table.

Loop Problem

Transparent switches work fine as long as there are no redundant switches in the sys tem. Systems administrators, however, like to have redundant switches (more than one switch between a pair of LANs) to make the system more reliable. If a switch fail, another switch takes over until the failed one is repaired or replaced. Redundancy can create loops in the system, which is very undesirable.

v. Router:

A router is three layer device; physical, data link and network layer. As a physical layer device it can regenerates bit, as data link layer device the router can check the physical addresses contained in a packet and as a network layer device it can check the network layer addresses (addresses in IP layer).
Routers are inter-networking devices as it can connects LAN-LAN, LAN-WAN, WAN-WAN.
Routers can change the physical addresses of a packet. For example, considering a packet is getting sent from LAN 1 to LAN 2 as in the above given figure. At this moment the source address is of the sender's address and the destination address is of a host of the LAN 2. Now when the packet reaches the router, the router changes the source address of the packet to its own address and then send the packet to the destination address.



GATEWAY

A gateway, as the name suggests, is a passage to connect two networks together that may work upon different networking models.

They basically work as the messenger agents that take data from one system, interpret it, and transfer it to another system.

Gateways are also called protocol converters and can operate at any network layer.

Gateways are generally more complex than switch or router.

it is also used in Application layer