Unit-2

The data link layer deals with algorithms for achieving reliable, efficient communication of whole units of information called frames (rather than individual bits, as in the physical layer) between two adjacent machines. By adjacent, we mean that the two machines are connected by a communication channel that acts conceptually like a wire (e.g., a coaxial cable, telephone line, or wireless channel). The essential property of a channel that makes it ‘‘wire-like’’ is that the bits are delivered in exactly the same order in which they are sent.

Machine A just puts the bits on the wire, and machine B just takes them off. Unfortunately, communication channels make errors occasionally. Furthermore, they have only a finite data rate, and there is a nonzero propagation delay between the time a bit is sent and the time it is received. These limitations have important implications for the efficiency of the data transfer. The protocols used for communications must take all these factors into consideration.

Data Link Layer Design Issues

The data link layer uses the services of the physical layer to send and receive bits over communication channels. It has a number of functions, including:

1. Providing a well-defined service interface to the network layer.

2. Dealing with transmission errors.

3. Regulating the flow of data so that slow receivers are not swamped by fast senders.

To accomplish these goals, the data link layer takes the packets it gets from the network layer and encapsulates them into frames for transmission. Each frame contains a frame header, a payload field for holding the packet, and a frame trailer, as illustrated in Fig. 3-1. Frame management forms the heart of what the data link layer does.

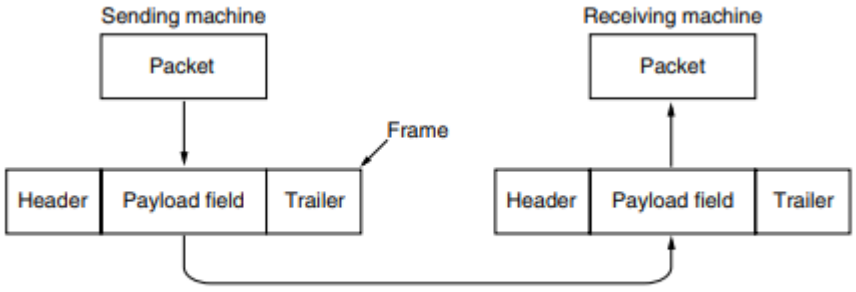


Figure 3-1. Relationship between packets and frames.

Services Provided to the Network Layer

The data link layer can be designed to offer various services. The actual services that are offered vary from protocol to protocol. Three reasonable possibilities that we will consider in turn are:

1. Unacknowledged connectionless service.

Unacknowledged connectionless service consists of having the source machine send independent frames to the destination machine without having the destination machine acknowledge them. Ethernet is a good example of a data link layer that provides this class of service. No logical connection is established beforehand or released afterward. If a

frame is lost due to noise on the line, n attempt is made to detect the loss or recover from it in the data link layer. This class of service is appropriate when the error rate is very low, so recovery is left to higher layers. It is also appropriate for real-time traffic, such as voice, in which late data are worse than bad data.

2. Acknowledged connectionless service.

When this service is offered, there are still no logical connections used, but each frame sent is individually acknowledged. In this way, the sender knows whether a frame has arrived correctly or been lost. If it has not arrived within a specified time interval, it can be sent again. This service is useful over unreliable channels, such as wireless systems. 802.11 (WiFi) is a good example of this class of service.

3. Acknowledged connection-oriented service.

With this service, the source and destination machines establish a connection before any data are transferred. Each frame sent over the connection is numbered, and the data link layer guarantees that each frame sent is indeed received. Furthermore, it guarantees that each frame is received exactly once and that all frames are received in the right order. Connection-oriented service thus provides the network layer processes with the equivalent of a reliable bit stream. It is appropriate over long, unreliable links such as a satellite channel or a long-distance telephone circuit.

Framing

The usual approach is for the data link layer to break up the bit stream into discrete frames, compute a short token called a checksum for each frame, and include the checksum in the frame when it is transmitted. (Checksum algorithms will be discussed later in this chapter.) 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 up the bit stream into frames is more difficult than it at first appears. A good design must make it easy for a receiver to find the start of new frames while using little of the channel bandwidth.

1. Byte count.

The first framing method uses a field in the header to specify the number of bytes in the frame. When the data link layer at the destination sees the byte count, it knows how many bytes follow and hence where the end of the frame is. This technique is shown in Fig. 3-3(a) for four small example frames of sizes 5, 5, 8, and 8 bytes, respectively.

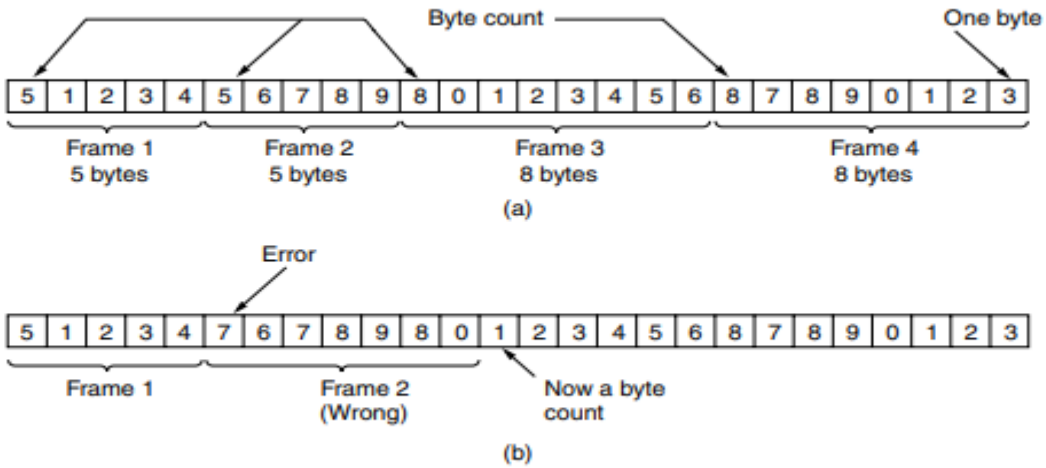


Figure 3-3. A byte stream. (a) Without errors. (b) With one error.

The trouble with this algorithm is that the count can be garbled by a transmission error. For example, if the byte counts of 5 in the second frame of Fig. 3-3(b) becomes a 7 due to a single bit flip, the destination will get out of synchronization. It will then be unable to locate the correct 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 bytes to skip over to get to the start of the retransmission. For this reason, the byte count method is rarely used by itself.

2. Flag bytes with byte stuffing.

Flag bytes with byte stuffing 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. (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.

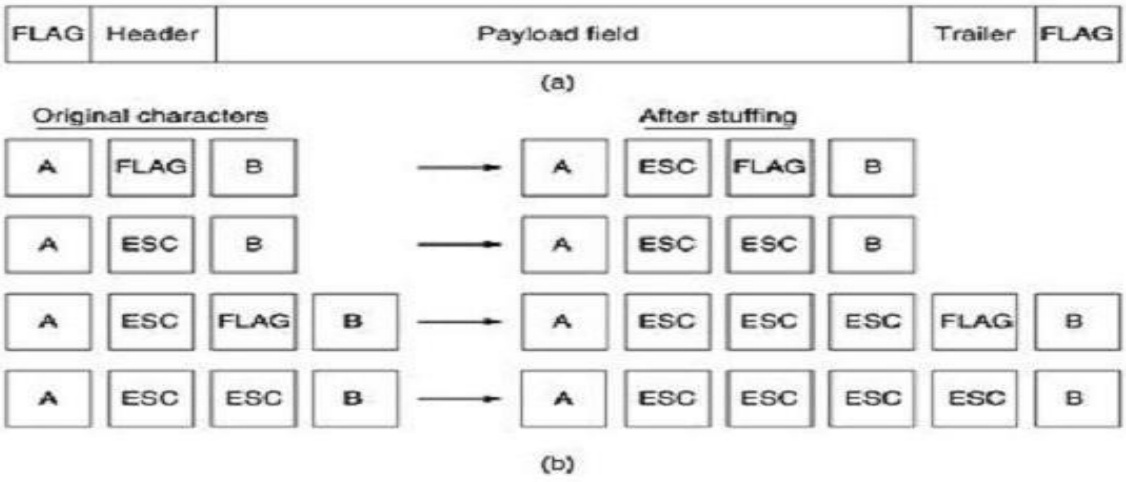


Fig: (a) A frame delimited by flag bytes (b) Four examples of byte sequences before and after byte stuffing

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.

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. (b). In all cases, the byte sequence delivered after de stuffing is exactly the same as the original byte sequence.

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, so a new technique had to be developed to allow arbitrary sized characters

3. Flag bits with bit stuffing.

Starting and ending flags, with bit stuffing 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.

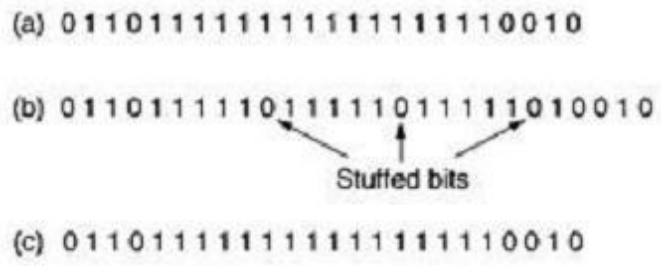
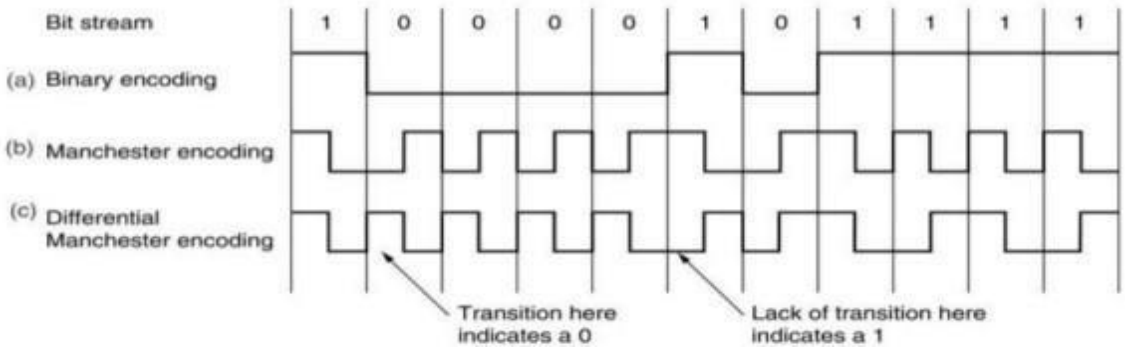


Fig: Bit stuffing. (a) The original data. (b) The data as they appear on the line. (c) The data as they are stored in the receiver's memory after destuffing.

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.

4. Physical layer coding violations

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



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

ERROR CONTROL

- Error control is concerned with insuring that all frames are eventually delivered (possibly in order) to a destination. How? Three items are required.

- **Acknowledgements:** Typically, reliable delivery is achieved using the "acknowledgments with retransmission" paradigm, whereby the receiver returns a special acknowledgment (ACK) frame to the sender indicating the correct receipt of a frame.
  - In some systems, the receiver also returns a negative acknowledgment (NACK) for incorrectly-received frames. This is nothing more than a hint to the sender so that it can retransmit a frame right away without waiting for a timer to expire.
- **Timers:** One problem that simple ACK/NACK schemes fail to address is recovering from a frame that is lost, and as a result, fails to solicit an ACK or NACK. What happens if an ACK or NACK becomes lost?
  - Retransmission timers are used to resend frames that don't produce an ACK. When sending a frame, schedule a timer to expire at some time after the ACK should have been returned. If the timer goes o, retransmit the frame.
- **Sequence Numbers:** Retransmissions introduce the possibility of duplicate frames. To suppress duplicates, add sequence numbers to each frame, so that a receiver can distinguish between new frames and old copies.

19

FLOW CONTROL

- *Flow control* deals with throttling the speed of the sender to match that of the receiver.

- Two Approaches:
  - **feedback-based flow control**, the receiver sends back information to the sender giving it permission to send more data or at least telling the sender how the receiver is doing
  - **rate-based flow control**, the protocol has a built-in mechanism that limits the rate at which senders may transmit data, without using feedback from the receiver.

- Various Flow Control schemes uses a common protocol that contains well-defined rules about when a sender may transmit the next frame. These rules often prohibit frames from being sent until the receiver has granted permission, either implicitly or explicitly.

20

ERROR CORRECTION AND DETECTION

- It is physically impossible for any data recording or transmission medium to be 100% perfect 100% of the time over its entire expected useful life.
  - In data communication, line noise is a fact of life (e.g., signal attenuation, natural phenomenon such as lightning, and the telephone repairman).
- As more bits are packed onto a square centimeter of disk storage, as communications transmission speeds increase, the likelihood of error increases-- sometimes geometrically.
- Thus, error detection and correction is critical to accurate data transmission, storage and retrieval.

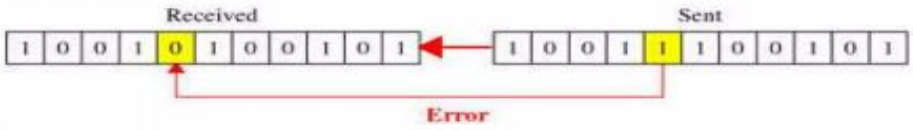
- Detecting and correcting errors requires *redundancy* -- sending additional information along with the data.

21

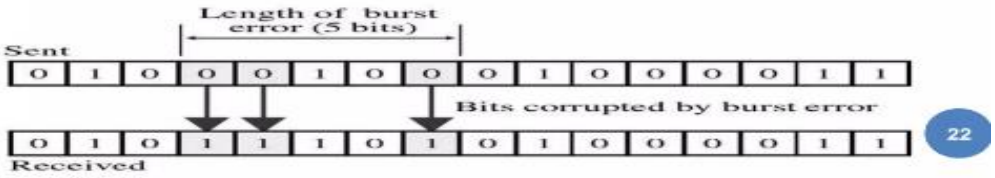
TYPES OF ERRORS

- There are two main types of errors in transmissions:

1. **Single bit error** : It means only one bit of data unit is changed from 1 to 0 or from 0 to 1.



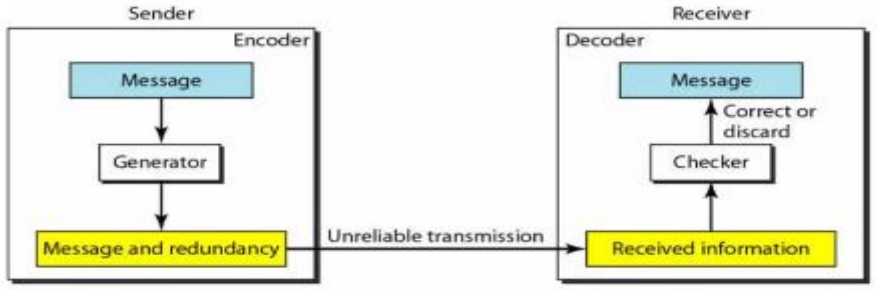
2. **Burst error** : It means two or more bits in data unit are changed from 1 to 0 from 0 to 1. In burst error, it is not necessary that only consecutive bits are changed. The length of burst error is measured from first changed bit to last changed bit



22

ERROR DETECTION Vs ERROR CORRECTION

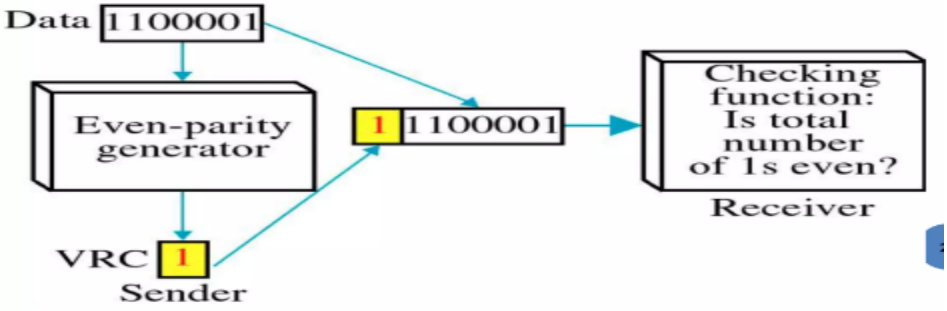
- There are two types of attacks against errors:
- **Error Detecting Codes:** Include enough redundancy bits to *detect* errors and use ACKs and retransmissions to recover from the errors.
- **Error Correcting Codes:** Include enough redundancy to detect *and* correct errors. The use of error-correcting codes is often referred to as forward error correction.



23

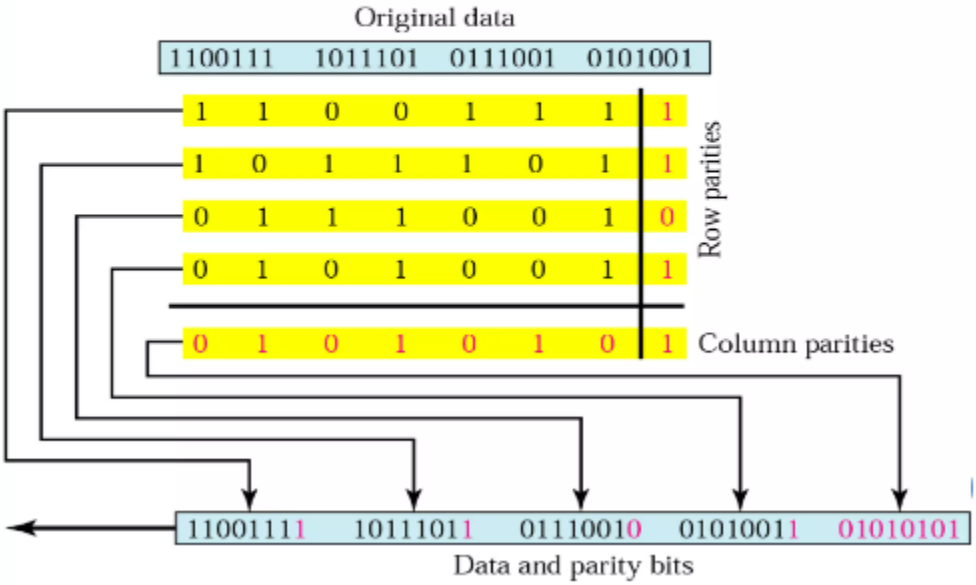
VERTICAL REDUNDANCY CHECK (VRC)

- Append a single bit at the end of data block such that the number of ones is even
  → Even Parity (odd parity is similar)
  0110011 → 01100110
  0110001 → 01100011
- VRC is also known as **Parity Check.** Detects **all odd-number errors** in a data block



TWO DIMENSIONAL PARITY CHECK

- Upon receipt, each character is checked according to its VRC parity value and then the entire block of characters is verified using the LRC block check character.



CRC (Already taken in Class)

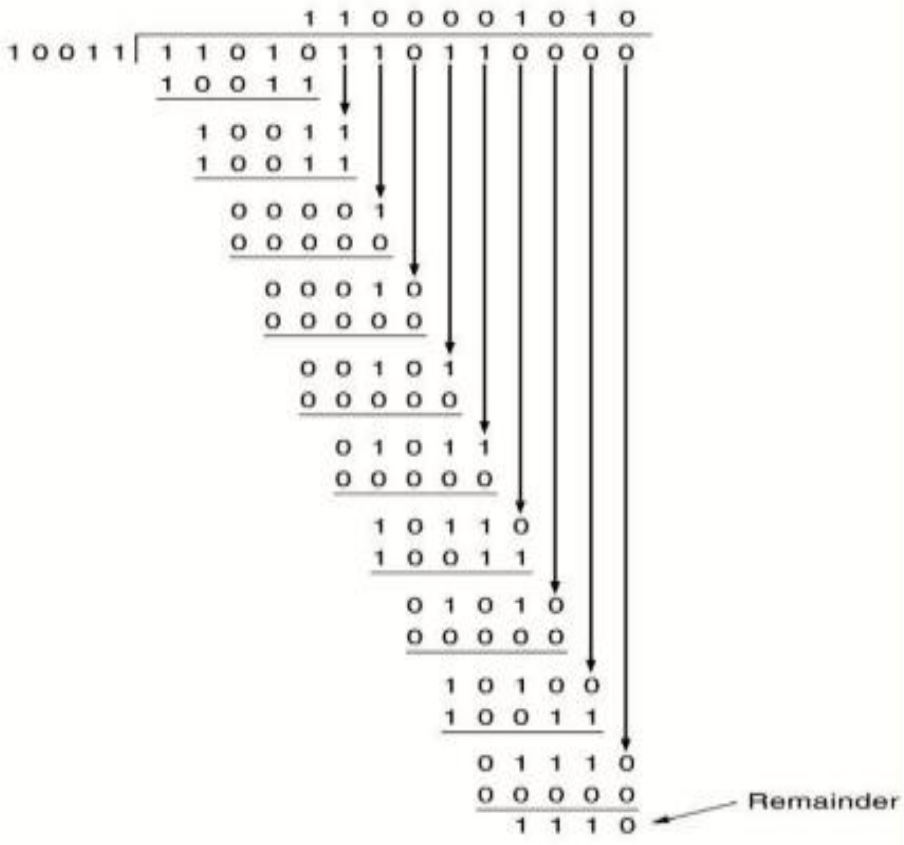
CYCLIC REDUNDANCY CHECK (CRC)

- The cyclic redundancy check, or CRC, is a technique for detecting errors in digital data, but not for making corrections when errors are detected. It is used primarily in data transmission
- In the CRC method, a certain number of check bits, often called a checksum, are appended to the message being transmitted. The receiver can determine whether or not the check bits agree with the data, to ascertain with a certain degree of probability whether or not an error occurred in transmission
- The CRC is based on polynomial arithmetic, in particular, on computing the remainder of dividing one polynomial in GF(2) (Galois field with two elements) by another.
- Can be easily implemented with small amount of hardware
  - Shift registers
  - XOR (for addition and subtraction)

GENERATOR POLYNOMIAL

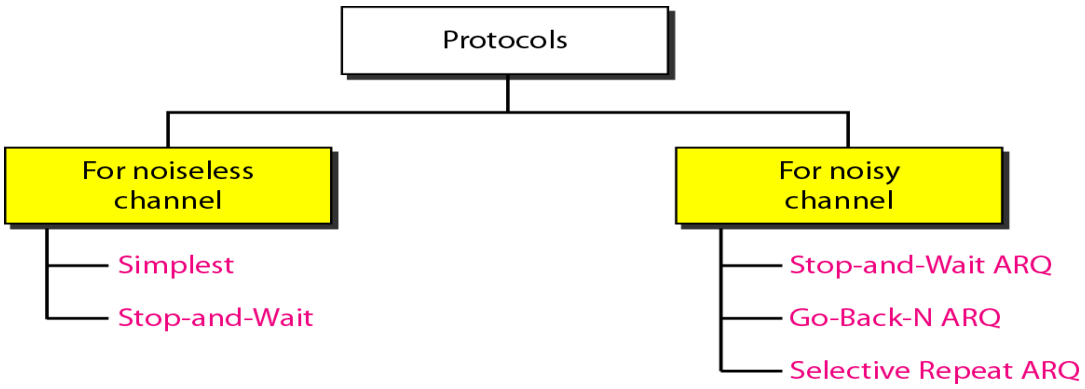
- A cyclic redundancy check (CRC) is a non-secure hash function designed to detect accidental changes to raw computer data, and is commonly used in digital networks and storage devices such as hard disk devices.
- CRCs are so called because the check (data verification) code is a redundancy (it adds zero information) and the algorithm is based on cyclic codes.
- The term CRC may refer to the check code or to the function that calculates it, which accepts data streams of any length as input but always outputs a fixed-length code
- The divisor in a cyclic code is normally called the **generator polynomial** or simply the generator. The proper
  - 1. It should have at least two terms.
  - 2. The coefficient of the term x^0 should be 1.
  - 3. It should not divide $x^t + 1$, for t between 2 and $n - 1$.
  - 4. It should have the factor $x + 1$.

Frame : 1 1 0 1 0 1 1 0 1 1
Generator: 1 0 0 1 1
Message after 4 zero bits are appended: 1 1 0 1 0 1 1 0 1 1 0 0 0 0

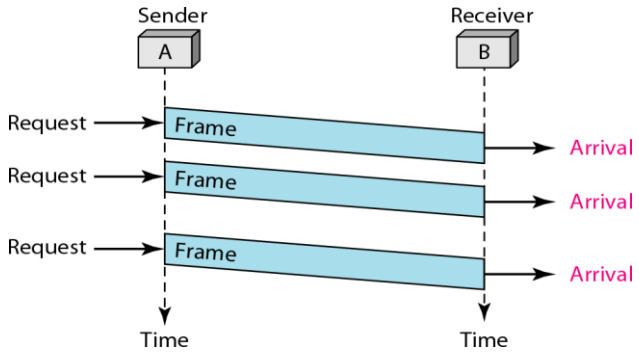


Transmitted frame: 1 1 0 1 0 1 1 0 1 1 1 1 1 0

ELEMENTARY DATA LINK PROTOCOLS

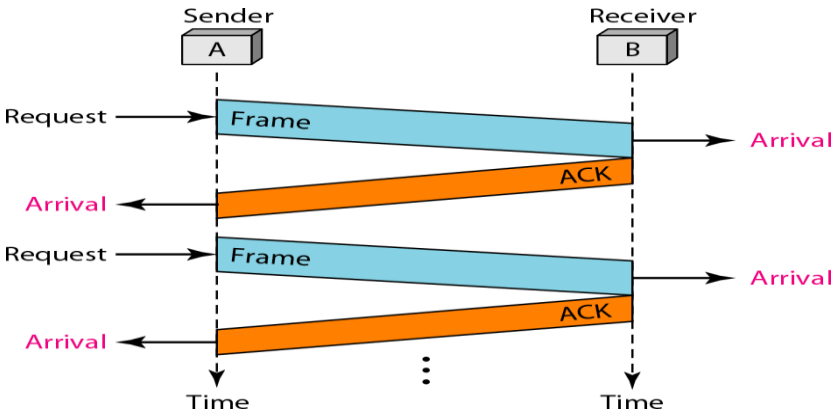


Simplest Protocol



It is very simple. The sender sends a sequence of frames without even thinking about the receiver. Data are transmitted in one direction only. Both sender & receiver 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 utopia protocol is unrealistic because it does not handle either flow control or error correction

Stop-and-wait Protocol



It is still very simple. The sender sends one frame and waits for feedback from the receiver. When the ACK arrives, the sender sends the next frame

It is Stop-and-Wait Protocol because the sender sends one frame, stops until it receives confirmation from the receiver (okay to go ahead), and then sends the next frame. We still have unidirectional communication for data

frames, but auxiliary ACK frames (simple tokens of acknowledgment) travel from the other direction. We add flow control to our previous protocol.

NOISY CHANNELS

Although the Stop-and-Wait Protocol gives us an idea of how to add flow control to its predecessor, noiseless channels are nonexistent. We can ignore the error (as we sometimes do), or we need to add error control to our protocols. We discuss three protocols in this section that use error control.
Sliding Window Protocols:

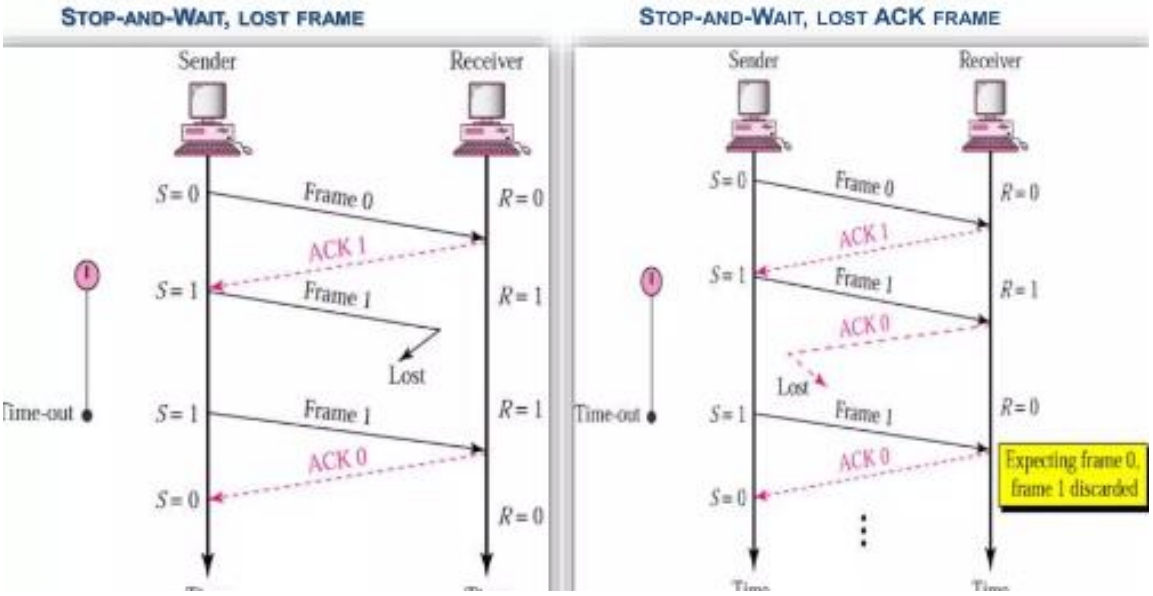
1. Stop-and-Wait Automatic Repeat Request
2. Go-Back-N Automatic Repeat Request
3. Selective Repeat Automatic Repeat Request

1 Stop-and-Wait Automatic Repeat Request

A SIMPLEX PROTOCOL FOR A NOISY CHANNEL

- In this protocol the unreal "error free" assumption in protocol 2 is dropped. Frames may be either damaged or lost completely. We assume that transmission errors in the frame are detected by the hardware checksum. One suggestion is that the sender would send a frame, the receiver would send an ACK frame only if the frame is received correctly. If the frame is in error the receiver simply ignores it; the transmitter would time out and would retransmit it.
- One fatal flaw with the above scheme is that if the ACK frame is lost or damaged, duplicate frames are accepted at the receiver without the receiver knowing it.

- Imagine a situation where the receiver has just sent an ACK frame back to the sender saying that it correctly received and already passed a frame to its host. However, the ACK frame gets lost completely, the sender times out and retransmits the frame. There is no way for the receiver to tell whether this frame is a retransmitted frame or a new frame, so the receiver accepts this duplicate happily and transfers it to the host. The protocol thus fails in this aspect.



- To overcome this problem it is required that the receiver be able to distinguish a frame that it is seeing for the first time from a retransmission. One way to achieve this is to have the sender put a **sequence number** in the header of each frame it sends. The receiver then can check the sequence number of each arriving frame to see if it is a new frame or a duplicate to be discarded.

- The receiver needs to distinguish only 2 possibilities: a new frame or a duplicate; a **1-bit sequence number** is sufficient. At any instant the receiver expects a particular sequence number. Any wrong sequence numbered frame arriving at the receiver is rejected as a duplicate. A correctly numbered frame arriving at the receiver is accepted, passed to the host, and the expected sequence number is incremented by 1 (modulo 2).

67

PIGGYBACKING

- Temporarily delaying transmission of outgoing acknowledgement so that they can be hooked onto the next outgoing data frame
- Advantage: higher channel bandwidth utilization
- Complication:
  - How long to wait for a packet to piggyback?
  - If longer than sender timeout period then sender retransmits
    ⇒ Purpose of acknowledgement is lost
- Solution for timing complexion
  - If a new packet arrives quickly
    ⇒ Piggybacking
  - If no new packet arrives after a receiver ack timeout
    ⇒ Sending a separate acknowledgement frame

73

SLIDING WINDOW PROTOCOLS

- The next three protocols are bidirectional protocols that belong to a class called sliding window protocols. (max sending window size, receiving window size)
  - One-bit sliding window protocol (1, 1)
  - Go back N (>1, 1)
  - Selective repeat (>1, >1)
- The three differ among themselves in terms of efficiency, complexity, and buffer requirements.
- Each outbound frame contains an n-bit sequence number
  - Range: 0 - MAX_SEQ (MAX_SEQ = 2^n - 1)
  - For stop-and-wait, n = 1 restricting the sequence numbers to 0 and 1 only

74

SENDING & RECEIVING WINDOWS

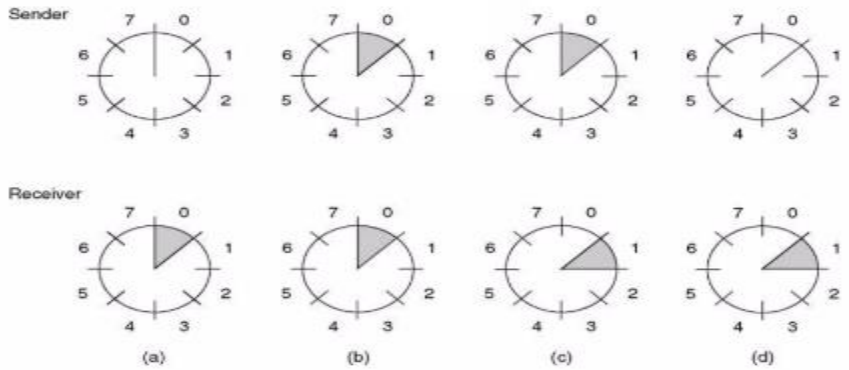
- At any instance of time
  - Sender maintains a set of sequence numbers of frames *permitted to send*
    - These frames fall within *sending window*
  - Receiver maintains a set of sequence numbers of frames *permitted to accept*
    - These frames fall within *receiving window*
- Lower limit, upper limit, and size of two windows *need not be the same* - Fixed or variable size
- **Senders Window** contains frames can be sent or have been sent but not yet acknowledged – *outstanding* frames
- When a packet arrives from network layer
  - Next highest sequence number assigned
  - Upper edge of window advanced by 1
- When an acknowledgement arrives
  - Lower edge of window advanced by 1

- 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.

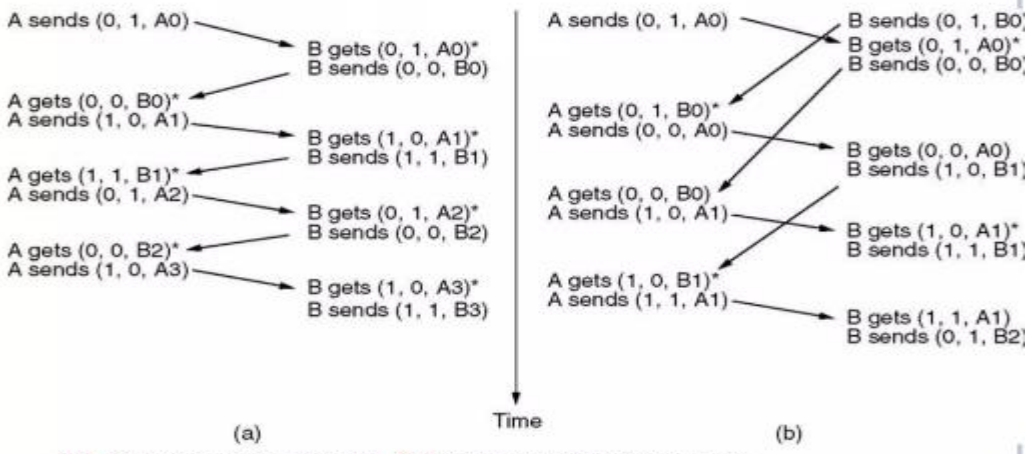
- Unlike the sender's window, the receiver's window always remains at its initial size.

A ONE BIT SLIDING WINDOW PROTOCOL



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.

A One Bit Sliding Window Protocol



(a) Case 1: Normal case. (b) Case 7: Abnormal case.
The notation is **(seq, ack, packet number)**. An asterisk indicates where a network layer accepts a packet.

82

Performance of Stop-and-Wait Protocol

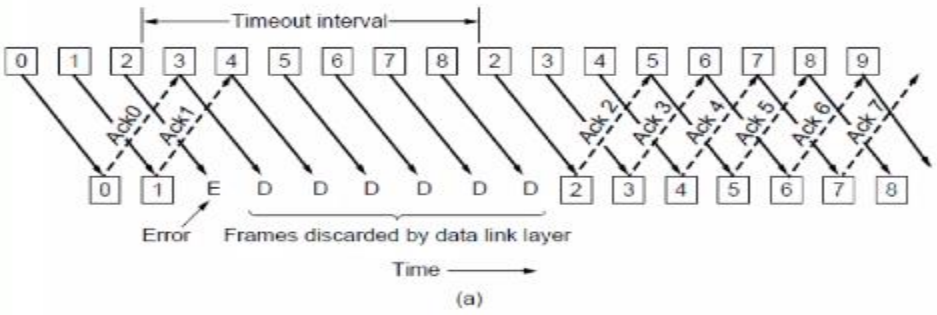
- In stop-and-wait, at any point in time, there is only one frame that is sent and waiting to be acknowledged.
- This is not a good use of transmission medium.
- To improve efficiency, multiple frames should be in transition while waiting for ACK.
- Solution: **PIPELINING**
  - Allowing *w* frames sent before blocking
- Problem: errors
- Solutions
  - **Go back *n* protocol (GNP)**
  - **Selective repeat protocol (SRP)**

Acknowledge *n* means frames *n*, *n*-1, *n*-2,... are acknowledged (i.e., received correctly)

Go Back *N* Protocol

- Improves efficiency of Stop and Wait by not waiting
- Keep Channel busy by continuing to send frames
- Allow a window of upto W_S outstanding frames
- Use m-bit sequence numbering
- Receiver discards all subsequent frames following an error one, and send no acknowledgement for those discarded
- Receiving window size = 1 (i.e., frames must be accepted in the order they were sent)
- Sending window might get full
  - If so, re-transmitting unacknowledged frames
- Wasting a lot of bandwidth if error rate is high

GO BACK N PROTOCOL



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.

SELECT REPEAT PROTOCOL

- Receiver stores correct frames following the bad one
- Sender retransmits the bad one after noticing
- Receiver passes data to network layer and acknowledge with the highest number
- Receiving window > 1 (i.e., any frame within the window may be accepted and buffered until all the preceding one passed to the network layer. Might need large memory
- ACK for frame *n* implicitly acknowledges all frames ≤ *n*
- SRP is often combined with NAK
- When error is *suspected* by receiver, receiver request retransmission of a frame
  - Arrival of a damaged frame
  - Arrival of a frame other than the expected\
- NAKs stimulate retransmission before the corresponding timer expires and thus improve performance.

SELECTIVE REPEAT WITH NAK