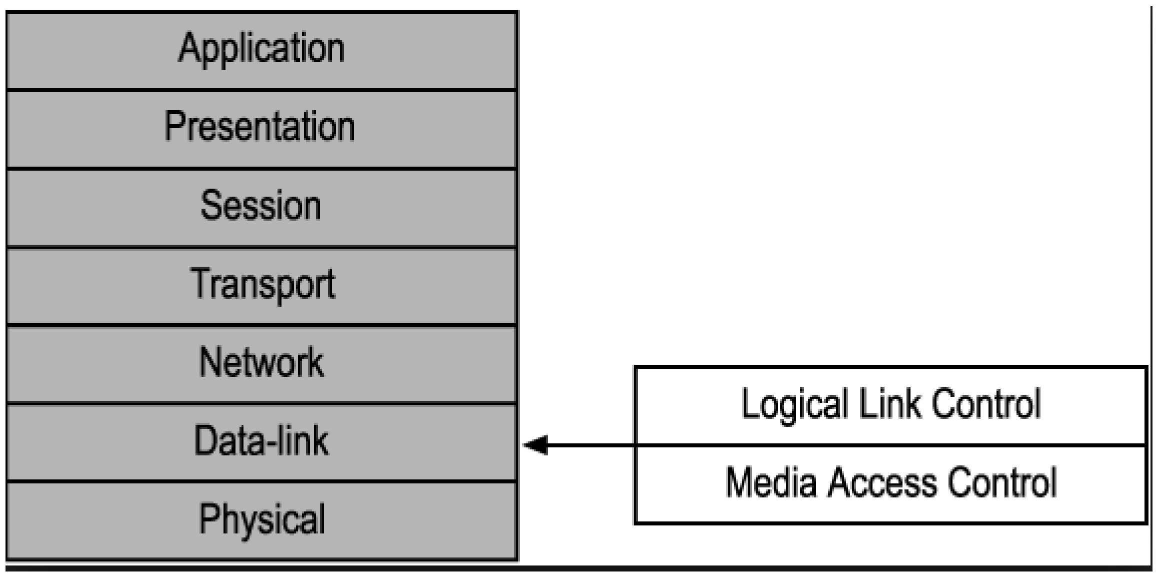
COMPUTER NETWORK
UNIT-II
Lecture-1

DATA LINK LAYER: INTRODUCTION
[RGPV June 2014]

Data Link Layer is second layer of OSI Layered Model. This layer is one of the most complicated layers and has complex functionalities and liabilities. Data link layer hides the details of underlying hardware and represents itself to upper layer as the medium to communicate.

Data link layer works between two hosts which are directly connected in some sense. This direct connection could be point to point or broadcast. Systems on broadcast network are said to be on same link. The work of data link layer tends to get more complex when it is dealing with multiple hosts on single collision domain.

Data link layer is responsible for converting data stream to signals bit by bit and to send that over the underlying hardware. At the receiving end, Data link layer picks up data from hardware which are in the form of electrical signals, assembles them in a recognizable frame format, and hands over to upper layer.



Data link layer has two sub-layers:

- Logical Link Control: It deals with protocols, flow-control, and error control
- Media Access Control**:** It deals with actual control of media

DATA LINK LAYER: SERVICES

- Encapsulation of network layer data packets into frames
- Frame synchronization
- Logical link control (LLC) sublayer:
  - Error control (automatic repeat request,ARQ), in addition to ARQ provided by

some transport-layer protocols, to forward error correction (FEC) techniques provided on thephysical layer, and to error-detection and packet cancelling provided at all layers, including the network layer. Data-link-layer error control (i.e. retransmission of erroneous packets) is provided in wireless networks and V.42 telephone network modems, but not in LAN protocols such as Ethernet, since bit errors are so uncommon in short wires. In that case, only error detection and canceling of erroneous packets are provided.

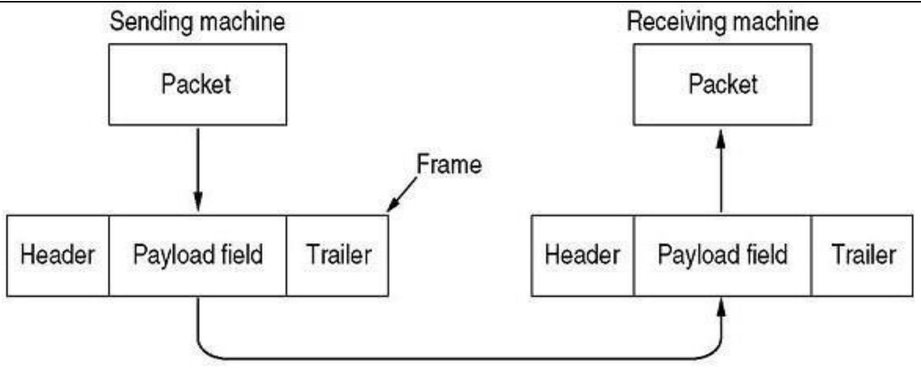
  - Flow control, in addition to the one provided on the transport layer. Data-link-layer error control is not used in LAN protocols such as Ethernet, but in modems and wireless networks.

- Media access control (MAC) sublayer:
  - Multiple access protocols for channel-access control, for example CSMA/CD protocols for collision detection and re-transmission in Ethernet bus networks and hub networks, or the CSMA/CA protocol for collision avoidance in wireless networks.
  - Physical addressing (MAC addressing)
  - LAN switching (packet switching) including MAC filtering and spanning tree protocol
  - Data packet queuing or scheduling
  - Store-and-forward switching or cut-through switching
  - Quality of Service (QoS) control
  - Virtual LANs (VLAN)

DATA LINK LAYER: FRAMING

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

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



Character Count
This method uses a field in the header to specify the number of characters in the frame. When the data link layer at the destination sees the character count,it knows how many characters follow, and hence where the end of the frame is. The disadvantage is that if the count is garbled by a transmission error, the destination will lose synchronization and will be unable to locate the start of the next frame. So, this method is rarely used.

Character stuffing
In the second method, each frame starts with the ASCII character sequence DLE STX and ends with the sequence DLE ETX.(where DLE is Data Link Escape, STX is Start of TeXt and ETX is End of TeXt.) This method overcomes the drawbacks of the character count method. If the destination ever loses synchronization, it only has to look for DLE STX and DLE ETX characters. If however, binary data is being transmitted then there exists a possibility of the characters DLE STX and DLE ETX occurring in the data. Since this can interfere with the framing, a technique called character stuffing is used. The sender's data link layer inserts an ASCII DLE character just before the DLE character in the data. The receiver's data link layer removes this DLE before this data is given to the network layer. However character stuffing is closely associated with 8-bit characters and this is a major hurdle in transmitting arbitrary sized characters.

Bit stuffing
[RGPV June 2013]

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

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

Lecture-2

DATALINK LAYER: FLOW CONTROL

Another important design issue that occurs in the data link layer (and higher layers as well) is what to do with a sender that systematically wants to transmit frames faster than the receiver can accept them. This situation can occur when the sender is running on a fast, powerful computer and the receiver is running on a slow, low-end machine. A common situation is when a smart phone requests a Web page from a far more powerful server, which then turns on the fire hose and blasts the data at the poor helpless phone until it is completely swamped. Even if the transmission is error free, the receiver may be unable to handle the frames as fast as they arrive and will lose some. Clearly, something has to be done to prevent this situation.

Two approaches are commonly used. In the first one, 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. In the second one, 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.

Feedback-based schemes are seen at both the link layer and higher layers. The latter is more common these days, in which case the link layer hardware is designed to run fast enough that it does not cause loss. For example, hardware implementations of the link layer as NICs (Network Interface Cards) are sometimes said to run at ‘‘wire speed,’’ meaning that they can handle frames as fast as they can arrive on the link. Any overruns are then not a link problem, so they are handled by higher layers.

Various feedback-based flow control schemes are known, but most of them use the same basic principle. The protocol 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.

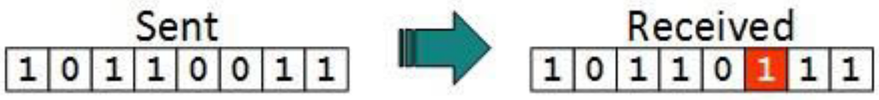
DATA LINK LAYER : ERROR CONTROL

There are many reasons such as noise, cross-talk etc., which may help data to get corrupted during transmission. The upper layers work on some generalized view of network architecture and are not aware of actual hardware data processing.Hence, the upper layers expect error-free transmission between the systems. Most of the applications would not function expectedly if they receive erroneous data. Applications such as voice and video may not be that affected and with some errors they may still function well.
Data-link layer uses some error control mechanism to ensure that frames (data bit streams) are transmitted with certain level of accuracy. But to understand how errors is controlled, it is essential to know what types of errors may occur.

Types of Errors

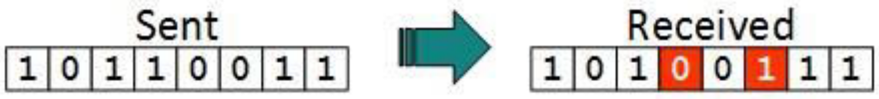
There may be three types of errors:

- **Single bit error**



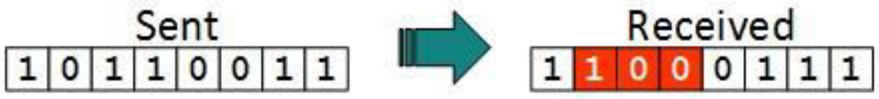
In a frame, there is only one bit, anywhere though, which is corrupt.

- **Multiple bits error**



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

- **Burst error**



Frame contains more than1 consecutive bits corrupted.

Error control mechanism may involve two possible ways:

Error detection

Error correction

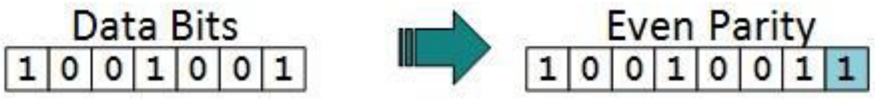
Error Detection

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

Parity Check

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

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

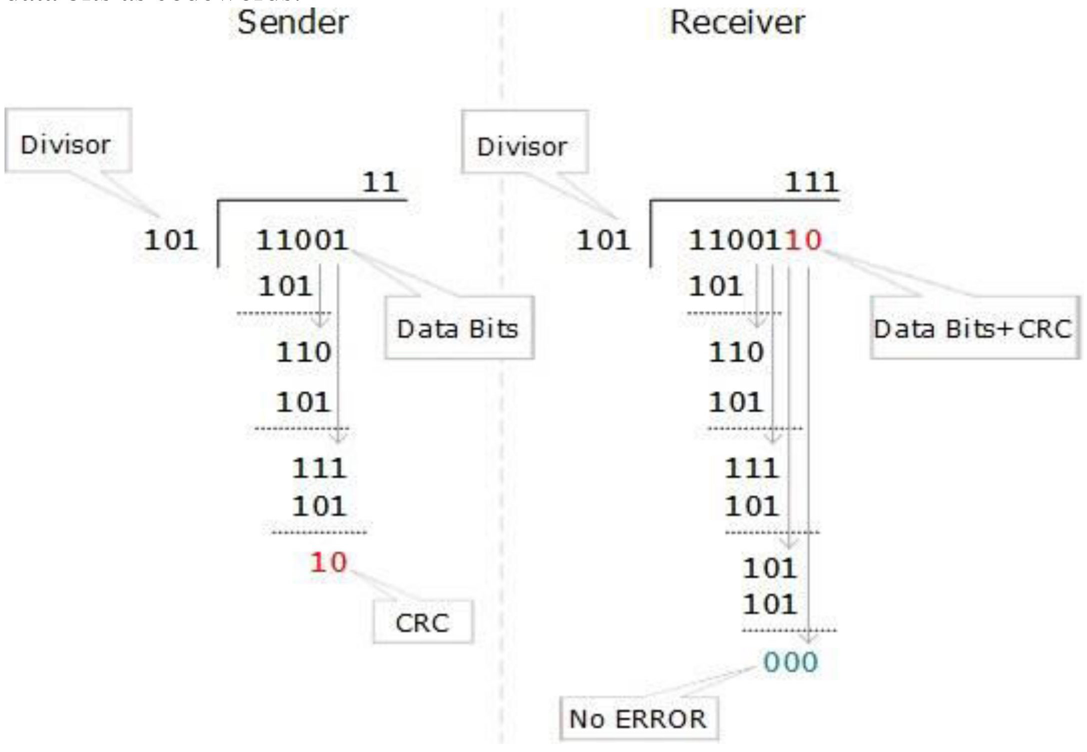


The receiver simply counts the number of 1s in a frame. If the count of 1s is even and even parity is used, the frame is considered to be not-corrupted and is accepted. If the count of 1s is odd and odd parity is used, the frame is still not corrupted.

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

Cyclic Redundancy Check (CRC)

CRC is a different approach to detect if the received frame contains valid data. This technique involves binary division of the data bits being sent. The divisor is generated using polynomials. The sender performs a division operation on the bits being sent and calculates the remainder. Before sending the actual bits, the sender adds the remainder at the end of the actual bits. Actual data bits plus the remainder is called a codeword. The sender transmits data bits as codewords.



At the other end, the receiver performs division operation on codewords using the same CRC divisor. If the remainder contains all zeros the data bits are accepted, otherwise it is considered as there some data corruption occurred in transit.

Error Correction

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

Backward Error Correction When the receiver detects an error in the data received, it requests back the sender to retransmit the data unit.

Forward Error Correction When the receiver detects some error in the data received, it executes error-correcting code, which helps it to auto-recover and to correct some kinds of errors.

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

To correct the error in data frame, the receiver must know exactly which bit in the frame is corrupted. To locate the bit in error, redundant bits are used as parity bits for error detection.For example, we take ASCII words (7 bits data), then there could be 8 kind of information we need: first seven bits to tell us which bit is error and one more bit to tell that there is no error.

For m data bits, r redundant bits are used. r bits can provide 2r combinations of information. In m+r bit codeword, there is possibility that the r bits themselves may get corrupted. So the number of r bits used must inform about m+r bit locations plus no-error information, i.e. m+r+1.

$$2^r >= m+r+1$$

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

Lecture-3

DATA LINK LAYER PROTOCOL

The basic function of the layer is to transmit frames over a physical communication link. Transmission may be *half duplex* or *full duplex*. To ensure that frames are delivered free of errors to the destination station (IMP) a number of requirements are placed on a data link protocol. The protocol (control mechanism) should be capable of performing:

1. The identification of a frame (i.e. recognise the first and last bits of a frame).
2. The transmission of frames of any length up to a given maximum. Any bit pattern is permitted in a frame.
3. The detection of transmission errors.
4. The retransmission of frames which were damaged by errors.
5. The assurance that no frames were lost.
6. In a multidrop configuration
    1. Some mechanism must be used for preventing conflicts caused by simultaneous transmission by many stations.
7. The detection of failure or abnormal situations for control and monitoring purposes.

It should be noted that as far as layer 2 is concerned a host message is pure data, every single bit of which is to be delivered to the other host. The frame header pertains to layer 2 and is never given to the host.

Elementary Data Link Protocols

An unrestricted simplex protocol
In this protocol:

- Data are transmitted in one direction only
- The transmitting (Tx) and receiving (Rx) hosts are always ready
- Processing time can be ignored
- Infinite buffer space is available
- No errors occur; i.e. no damaged frames and no lost frames (perfect channel).

A simplex stop-and-wait protocol
In this protocol we assume that

- Data are transmitted in one direction only
- No errors occur (perfect channel)
- The receiver can only process the received information at a finite rate

These assumptions imply that the transmitter cannot send frames at a rate faster than the receiver can process them.The problem here is how to prevent the sender from flooding the receiver.

A general solution to this problem is to have the receiver provide some sort of feedback to the sender. The process could be as follows: The receiver send an acknowledge frame back to the sender telling the sender that the last received frame has been processed and passed to the host; permission to send the next frame is granted. The sender, after having sent a frame, must wait for the acknowledge frame from the receiver before sending another frame. This protocol is known as stop-and-wait.

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).

Sliding window protocol

[RGPV June 2014]

Full duplex. A **s**liding window protocol is a feature of packet-based data transmission protocols. Sliding window protocols are used where reliable in-order delivery of packets is required, such as in the Data Link Layer (OSI model) as well as in the Transmission Control Protocol (TCP).

Conceptually, each portion of the transmission (packets in most data link layers, but bytes in TCP) is assigned a unique consecutive sequence number, and the receiver uses the numbers to place received packets in the correct order, discarding duplicate packets and identifying missing ones. The problem with this is that there is no limit on the size of the sequence number that can be required.

The simplest sliding window: stop-and-wait
[RGPV June 2014]

Although commonly distinguished from the sliding-window protocol, the stop-and-wait ARQ protocol is actually the simplest possible implementation of it. The transmit window is 1 packet, and the receive window is 1 packet. Thus, N=1+1=2 possible sequence numbers (conveniently represented by a single bit) are required.

Go-Back-N

Go-Back-N ARQ is the sliding window protocol with w_t>1, but a fixed w_r=1. The receiver refuses to accept any packet but the next one in sequence. If a packet is lost in transit, following packets are ignored until the missing packet is retransmitted, a minimum loss of one round trip time. For this reason, it is inefficient on links that suffer frequent packet loss

Selective repeat
[RGPV June 2013]

The most general case of the sliding window protocol is Selective Repeat ARQ. This requires a much more capable receiver, which can accept packets with sequence numbers higher than the current n_r and store them until the gap is filled in.

The advantage, however, is that it is not necessary to discard following correct data for one round-trip time before the transmitter can be informed that a retransmission is required. This is therefore preferred for links with low reliability and/or a high bandwidth-delay product.

The window size w_r need only be larger than the number of consecutive lost packets that can be tolerated. Thus, small values are popular; w_r=2 is common.

Lecture-4

HYBRID ARQ

Hybrid automatic repeat request (hybrid ARQ or HARQ) is a combination of high-rate forward error-correcting coding and ARQ error-control. In standard ARQ, redundant bits are added to data to be transmitted using an error-detecting (ED) code such as a cyclic redundancy check (CRC). Receivers detecting a corrupted message will request a new message from the sender.

In Hybrid ARQ, the original data is encoded with a forward error correction (FEC) code, and the parity bits are either immediately sent along with the message or only transmitted upon request when a receiver detects an erroneous message. The ED code may be omitted when a code is used that can perform both forward error correction (FEC) in addition to error detection, such as a Reed-Solomon code.

The FEC code is chosen to correct an expected subset of all errors that may occur, while the ARQ method is used as a fall-back to correct errors that are in correctable using only the redundancy sent in the initial transmission. As a result, hybrid ARQ performs better than ordinary ARQ in poor signal conditions, but in its simplest form this comes at the expense of significantly lower throughput in good signal conditions. There is typically a signal quality cross-over point below which simple hybrid ARQ is better, and above which basic ARQ is better.

The simplest version of HARQ, Type I HARQ, adds both ED and FEC information to each message prior to transmission. When the coded data block is received, the receiver first decodes the error-correction code. If the channel quality is good enough, all transmission errors should be correctable, and the receiver can obtain the correct data block. If the channel quality is bad, and not all transmission errors can be corrected, the receiver will detect this situation using the error-detection code, then the received coded data block is rejected and a re-transmission is requested by the receiver, similar to ARQ.

In a more sophisticated form, Type II HARQ, the message originator alternates between message bits along with error detecting parity bits and only FEC parity bits. When the first transmission is received error free, the FEC parity bits are never sent. Also, two consecutive transmissions can be combined for error correction if neither is error free.

BIT ORIENTED PROTOCOLS

A bit-oriented protocol is a communications protocol that sees the transmitted data as an *opaque* stream of bits with no semantics, or meaning. Control codes are defined in terms of bit sequences instead of characters. Bit oriented protocol can transfer data frames regardless of frame contents. It can also be stated as "bit stuffing" this technique allows the data frames to contain an arbitrary number of bits and allows character codes with arbitrary number of bits per character.

Synchronous framing High-Level Data Link Control is a popular bit-oriented protocol.

SDLC

Synchronous Data Link Control (SDLC) is a computer communications protocol. It is the layer 2 protocol for IBM's Systems Network Architecture (SNA). SDLC supports multipoint links as well as error correction. It also runs under the assumption that an SNA header is present after the SDLC header. SDLC was mainly used by IBM mainframe and midrange systems; however, implementations exist on many platforms from many vendors. The use of SDLC (and SNA) is becoming more and more rare, mostly replaced by IP-based protocols or being tunneled through IP (using AnyNet or other technologies).

SDLC operates independently on each communications link, and can operate on point-to-point multipoint or loop facilities, on switched or dedicated, two-wire or four-wire circuits, and with full-duplex and half-duplex operation. A unique characteristic of SDLC is its ability to mix half-duplex secondary stations with full-duplex primary stations on four-wire circuits, thus reducing the cost of dedicated facilities.

Intel used SDLC as a base protocol for BITBUS, still popular in Europe as fieldbus and included support in several controllers (i8044/i8344, i80152).

Lecture-5

HDLC
[RGPV June 2014]

High-Level Data Link Control (HDLC) is a bit-oriented code-transparent synchronous data link layer protocol developed by theInternational Organization for Standardization (ISO). The original ISO standards for HDLC are:

- ISO 3309 – Frame Structure
- ISO 4335 – Elements of Procedure
- ISO 6159 – Unbalanced Classes of Procedure
- ISO 6256 – Balanced Classes of Procedure

The current standard for HDLC is ISO 13239, which replaces all of those standards.

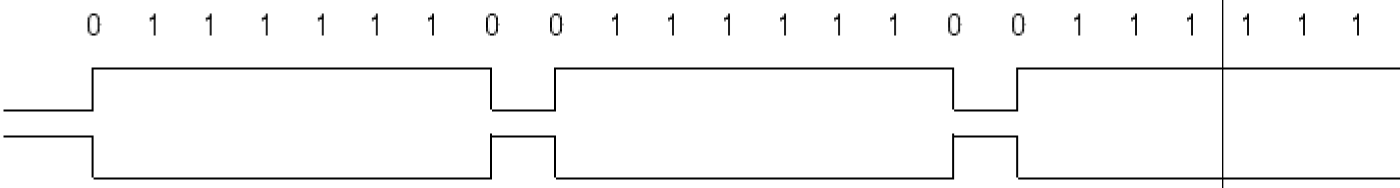
HDLC provides both connection-oriented and connectionless service.

HDLC can be used for point to multipoint connections, but is now used almost exclusively to connect one device to another, using what is known as Asynchronous Balanced Mode (ABM). The original master-slave modes Normal Response Mode (NRM) and Asynchronous Response Mode (ARM) are rarely used.

FRAMING

HDLC frames can be transmitted over synchronous or asynchronous serial communication links. Those links have no mechanism to mark the beginning or end of a frame, so the beginning and end of each frame has to be identified. This is done by using a frame delimiter, or *flag*, which is a unique sequence of bits that is guaranteed not to be seen inside a frame. This sequence is '01111110', or, in hexadecimal notation, 0x7E. Each frame begins and ends with a frame delimiter. A frame delimiter at the end of a frame may also mark the start of the next frame. A sequence of 7 or more consecutive 1-bits within a frame will cause the frame to be aborted.

When no frames are being transmitted on a simplex or full-duplex synchronous link, a frame delimiter is continuously transmitted on the link. Using the standard NRZI encoding from bits to line levels (0 bit = transition, 1 bit = no transition), this generates one of two continuous waveforms, depending on the initial state:



This is used by modems to train and synchronize their clocks via phase-locked loops. Some protocols allow the 0-bit at the end of a frame delimiter to be shared with the start of the next frame delimiter, i.e. '011111101111110'.

Frame structure

The contents of an HDLC frame are shown in the following table:

Flag	Address	Control	Information	FCS	Flag
8 bits	8 or more bits	8 or 16 bits	Variable length, 0 or more bits	16 or 32 bits	8 bits

Note that the end flag of one frame may be (but does not have to be) the beginning (start) flag of the next frame.

Data is usually sent in multiples of 8 bits, but only some variants require this; others theoretically permit data alignments on other than 8-bit boundaries.

The frame check sequence (FCS) is a 16-bit CRC-CCITT or a 32-bit CRC-32 computed over the Address, Control, and Information fields. It provides a means by which the receiver can detect errors that may have been induced during the transmission of the frame, such as lost bits, flipped bits, and extraneous bits. However, given that the algorithms used to calculate the FCS are such that the probability of certain types of transmission errors going undetected increases with the length of the data being checked for errors, the FCS can implicitly limit the practical size of the frame.

If the receiver's calculation of the FCS does not match that of the sender's, indicating that the frame contains errors, the receiver can either send a negative acknowledge packet to the sender, or send nothing. After either receiving a negative acknowledge packet or timing out waiting for a positive acknowledge packet, the sender can retransmit the failed frame.

The FCS was implemented because many early communication links had a relatively high bit error rate, and the FCS could readily be computed by simple, fast circuitry or software. More effective forward error correction schemes are now widely used by other protocols.

There are three fundamental types of HDLC frames.

- Information frames, or I-frames, transport user data from the network layer. In addition they can also include flow and error control information piggybacked on data.
- Supervisory Frames, or S-frames, are used for flow and error control whenever piggybacking is impossible or inappropriate, such as when a station does not have data to send. S-frames do not have information fields.
- Unnumbered frames, or U-frames, are used for various miscellaneous purposes, including link management. Some U-frames contain an information field, depending on the type.

Piggybacking

[RGPV June 2014], [RGPV June 2012]

Piggybacking is a bi-directional data transmission technique in the network layer (OSI model). It makes the most of the sent data frames from receiver to emitter, adding the confirmation that the data frame sent by the sender was received successfully (ACK acknowledge). This practically means, that instead of sending an acknowledgement in an individual frame it is piggy-backed on the data frame.

Piggybacking data is a bit different from Sliding Window Protocol used in the OSI model. In the data frame itself, we incorporate one additional field for acknowledgment (called ACK).

Whenever party A wants to send data to party B, it will send the data along with this ACK field. Considering the sliding window here of size 8 bits, if A has received frames up to 5 correctly (from B), and wants to send frames starting from frame 6, it will send ACK6 with the data.

Three rules govern the piggybacking data transfer.

- If station A wants to send both data and an acknowledgment, it keeps both fields there.
- If station A wants to send just the acknowledgment, then a separate ACK frame is sent.
- If station A wants to send just the data, then the last acknowledgment field is sent along with the data. Station B simply ignores this duplicate ACK frame upon receiving.

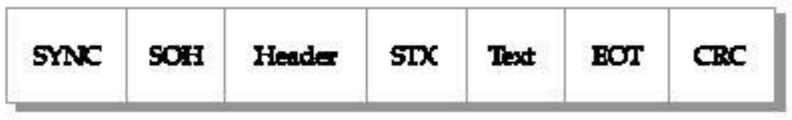
Lecture-6

BISYNC

Binary Synchronous Communication (BSC or Bisync) is an IBM character-oriented, half-duplex link protocol, announced in 1967 after the introduction of System/360. It replaced the synchronous transmit-receive (STR) protocol used with second generation computers. The intent was that common link management rules could be used with three different character encodings for messages. Six-bit Transcode looked backwards to older systems.

BISYNC establishes rules for transmitting binary-coded data between a terminal and a host computer's BISYNC port. While BISYNC is a half-duplex protocol, it will synchronize in both directions on a full-duplex channel. BISYNC supports both point-to-point (over leased or dial-up lines) and multipoint transmissions. Each message must be acknowledged, adding to its overhead.

BISYNC is character oriented, meaning that groups of bits (bytes) are the main elements of transmission, rather than a stream of bits. The BISYNC frame is pictured next. It starts with two sync characters that the receiver and transmitter use for synchronizing. This is followed by a start of header (SOH) command, and then the header. Following this are the start of text (STX) command and the text. Finally, an end of text (EOT) command and a cyclic redundancy check (CRC) end the frame. The CRC provides error detection and correction.



Most of the bisynchronous protocols, of which there are many, provide only half-duplex transmission and require an acknowledgment for every block of transmitted data. Some do provide full-duplex transmission and bit-oriented operation.

BISYNC has largely been replaced by the more powerful SDLC (Synchronous Data Link Control).

Bisync is an acronym shortened from "binary synchronous". Bisync is one of the names commonly used when referring to a synchronous communications protocol introduced by IBM back in 1964 with the introduction of a product called the 270X Transmission Control Unit.

The full name of the protocol is the "Binary Synchronous Communication" protocol. This is usually abbreviated to "BSC". Whenever you see the terms "bisync" and "BSC" in conjunction with communications protocols, they are referring to the same thing.

Furthermore, over the years, other terms have come into use and are often used interchangeably with bisync and BSC. For example, "3780 protocol", "3780 bisync", "2780 protocol", "2780 bisync", and "2780/3780 protocol". While technically these terms may be inaccurate, they do have a practical basis and convey meaning.

To be precise, 2780 and 3780 were model numbers of IBM remote job entry (RJE) data terminals -- namely the IBM 2780 Data Communications Terminal and the IBM 3780 Data Communications Terminal . These terminals used punch cards and consisted of a card reader, a card punch, and a line printer. These terminals used the bisync protocol to transmit and receive data with an IBM mainframe computer. Usually dial-up or leased telephone lines and 2400 baud Bell 201C, and then later via 4800 baud Bell 208A/B, modems were used to connect the terminal and mainframe.

Link control

The link control protocol is similar to STR. The designers attempted to protect against simple transmission errors. The protocol requires that every message be acknowledged (ACK0/ACK1) or negatively acknowledged (NAK), so transmission of small packets has high transmission overhead. The protocol can recover from a corrupted data frame, a lost data frame, and a lost acknowledgment.

Error recovery is by retransmission of the corrupted frame. Since Bisync data packets are not serial-numbered, it's considered possible for a data frame to go missing without the receiver realizing it. Therefore, alternating ACK0s and ACK1s are deployed; if the transmitter receives the wrong ACK, it can assume a data packet (or an ACK) went missing. A potential flaw is that corruption of ACK0 into ACK1 could result in duplication of a data frame.

Error protection for ACK0 and ACK1 is weak. The Hamming distance between the two messages is only two bits.

The protocol is half-duplex (2-wire). In this environment, packets or frames of transmission are strictly unidirectional, necessitating 'turn-around' for even the simplest purposes, such as acknowledgments. Turn-around involves

- the reversal of transmission direction,
- quiescing of line echo,
- resyncing.

In a 2-wire environment, this causes a noticeable round-trip delay and reduces performance.

Some datasets support full-duplex operation, and full-duplex (4-wire) can be used in many circumstances to improve performance by eliminating the turn-around time, at the added expense of 4-wire installation and support. In typical full-duplex, data packets are transmitted along one wire pair while the acknowledgements are returned along the other.

Lecture-7

LAP AND LAPB

Link Access Procedure (LAP) protocols are Data Link layer protocols for framing and transmitting data across point-to-point links. LAP was orginally derived from HDLC (High-Level Data Link Control), but was later updated and renamed LAPB (LAP Balanced).

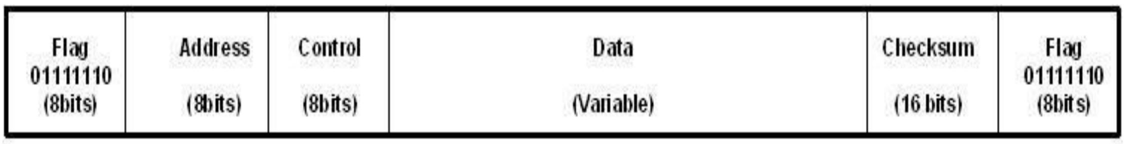
LAPB is the data link protocol for X.25.LAPB is a bit-oriented protocol derived from HDLC that ensures that frames are error free and in the right sequence. LAPB is specified in ITU-T Recommendation X.25 and ISO/IEC 7776. It can be used as a Data Link Layer protocol implementing the connection-mode data link service in the OSI Reference Model as defined by ITU-T Recommendation X.222.

LAPB is used to manage communication and packet framing between data terminal equipment (DTE) and the data circuit-terminating equipment (DCE) devices in the X.25 protocol stack. LAPB is essentially HDLC in Asynchronous Balanced Mode (ABM). LAPB sessions can be established by either the DTE or DCE. The station initiating the call is determined to be the primary, and the responding station is the secondary.

Frame types

- I-Frames (Information frames): Carries upper-layer information and some control information. I-frame functions include sequencing, flow control, and error detection and recovery. I-frames carry send and receive sequence numbers.
- S-Frames (Supervisory Frames): Carries control information. S-frame functions include requesting and suspending transmissions, reporting on status, and acknowledging the receipt of I-frames. S-frames carry only receive sequence numbers.
- U-Frames (Unnumbered Frames): carries control information. U-frame functions include link setup and disconnection, as well as error reporting. U-frames carry no sequence numbers
-

Frame format



Flag 01111110 (8bits)	Address (8bits)	Control (8bits)	Data (Variable)	Checksum (16 bits)	Flag 01111110 (8bits)

Flag – The value of the flag is always 0x7E. In order to ensure that the bit pattern of the frame delimiter flag does not appear in the data field of the frame (and therefore cause frame misalignment), a technique known as Bit stuffing is used by both the transmitter and the receiver.

Address field – In LAPB, this field has no meaning since the protocol works in a point to point mode and the DTE network address is represented in the layer 3 packets. This byte is therefore put to a different use; it separates the link commands from the responses and can have only two values: 0x01 and 0x03. 01 identifies frames containing commands from DTE to DCE and responses to these commands from DCE to DTE. 03 is used for frames containing commands from DCE to DTE and for responses from DTE to DCE. Therefore, one side must be configured as a Layer 2 DTE and the other as a Layer 2 DCE (you must not confuse this with the more familiar Layer 1 DCE and DTE designations).

Control field – it serves to identify the type of the frame. In addition, it includes sequence numbers, control features and error tracking according to the frame type.

Modes of operation

LAPB works in the Asynchronous Balanced Mode (ABM). This mode is balanced (i.e., no master/slave relationship) and is signified by the SABM(E)/SM frame. Each station may initialize, supervise, recover from errors, and send frames at any time. The DTE and DCE are treated as equals.

FCS – The Frame Check Sequence enables a high level of physical error control by allowing the integrity of the transmitted frame data to be checked.

Window size – LAPB supports an extended window size (modulo 128 and modulo 32768) where the maximum number of outstanding frames for acknowledgment is raised from 7 (modulo 8) to 127 (modulo 128) and 32767 (modulo 32768).

Protocol operation

LAPB has no master/slave node relationships. The sender uses the Poll bit in command frames to insist on an immediate response. In the response frame this same bit becomes the receivers Final bit. The receiver always turns on the Final bit in its response to a command from the sender with the Poll bit set. The P/F bit is generally used when either end becomes unsure about proper frame sequencing because of a possible missing acknowledgment, and it is necessary to re-establish a point of reference. It is also used to trigger an acknowledgment of outstanding I-frames.

Lecture-8

Protocol verification: Finite State Machine Models

[RGPV June 2013]

A **finite-state machine** (**FSM**) or **finite-state automaton** (plural: *automata*), or simply a **state machine**, is a mathematical model of computation used to design both computer programs and sequential logic circuits. It is conceived as an abstract machine that can be in one of a finite number of *states*. The machine is in only one state at a time; the state it is in at any given time is called the *current state*. It can change from one state to another when initiated by a triggering event or condition; this is called a *transition*. A particular FSM is defined by a list of its states, and the triggering condition for each transition.

The behavior of state machines can be observed in many devices in modern society which perform a predetermined sequence of actions depending on a sequence of events with which they are presented. Simple examples are vending machines which dispense products when the proper combination of coins is deposited, elevators which drop riders off at upper floors before going down, traffic lights which change sequence when cars are waiting, and combination locks which require the input of combination numbers in the proper order.

Finite-state machines can model a large number of problems, among which are electronic design automation, communication protocol design, language parsing and other engineering applications. In biology and artificial intelligence research, state machines or hierarchies of state machines have been used to describe neurological systems. In linguistics, they are used to describe simple parts of the grammars of natural languages.

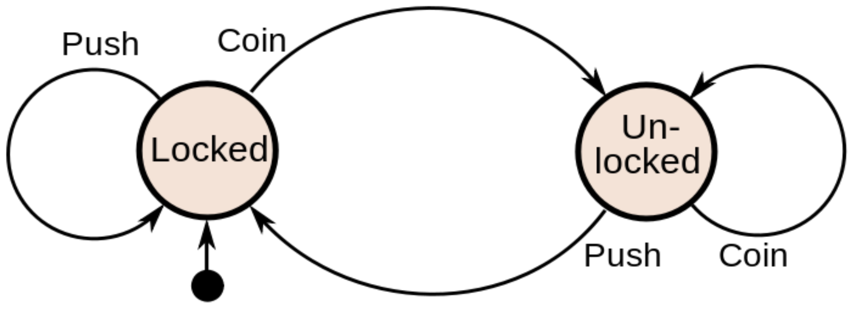
Considered as an abstract model of computation, the finite state machine is weak; it has less computational power than some other models of computation such as the Turing machine.That is, there are tasks which no FSM can do, but some Turing machines can. This is because the FSM has limited memory. The memory is limited by the number of states. FSMs are studied in the more general field of automata theory.

Example: a turnstile

An example of a very simple mechanism that can be modeled by a state machine is a turnstile. A turnstile, used to control access to subways and amusement park rides, is a gate with three rotating arms at waist height, one across the entryway. Initially the arms are locked, barring the entry, preventing customers from passing through. Depositing a coin or token in a slot on the turnstile unlocks the arms, allowing a single customer to push through. After the customer passes through, the arms are locked again until another coin is inserted.

Considered as a state machine, the turnstile has two states: ***Locked*** and ***Unlocked***. There are two inputs that affect its state: putting a coin in the slot (***coin***) and pushing the arm (***push***). In the locked state, pushing on the arm has no effect; no matter how many times the input ***push*** is given, it stays in the locked state. Putting a coin in – that is, giving the machine a ***coin*** input – shifts the state from ***Locked*** to ***Unlocked***. In the unlocked state, putting additional coins in has no effect; that is, giving additional ***coin*** inputs does not change the state. However, a customer pushing through the arms, giving a ***push*** input, shifts the state back to ***Locked***.

The turnstile state machine can be represented by a state transition table, showing for each state the new state and the output (action) resulting from each input



Concepts and terminology

A *state* is a description of the status of a system that is waiting to execute a *transition*. A transition is a set of actions to be executed when a condition is fulfilled or when an event is received. For example, when using an audio system to listen to the radio (the system is in the "radio" state), receiving a "next" stimulus results in moving to the next station. When the system is in the "CD" state, the "next" stimulus results in moving to the next track. Identical stimuli trigger different actions depending on the current state.

In some finite-state machine representations, it is also possible to associate actions with a state:

- Entry action: performed *when entering* the state,
- Exit action: performed *when exiting* the state.

Petri net

A Petri net (also known as a place/transition net or P/T net) is one of several mathematical modeling languages for the description of distributed systems. A Petri net is a directed bipartite graph, in which the nodes represent transitions (i.e. events that may occur, signified by bars) and places (i.e. conditions, signified by circles). The directed arcs describe which places are pre- and/or postconditions for which transitions (signified by arrows). Some sources state that Petri nets were invented in August 1939 by Carl Adam Petri — at the age of 13 — for the purpose of describing chemical processes.

Like industry standards such as UML activity diagrams, BPMN and EPCs, Petri nets offer a graphical notation for stepwise processes that include choice, iteration, and concurrent execution. Unlike these standards, Petri nets have an exact mathematical definition of their execution semantics, with a well-developed mathematical theory for process analysis.

Petri net basics

A Petri net consists of *places*, *transitions*, and *arcs*. Arcs run from a place to a transition or vice versa, never between places or between transitions. The places from which an arc runs to a transition are called the *input places* of the transition; the places to which arcs run from a transition are called the *output places* of the transition.

Graphically, places in a Petri net may contain a discrete number of marks called *tokens*. Any distribution of tokens over the places will represent a configuration of the net called a *marking*. In an abstract sense relating to a Petri net diagram, a transition of a Petri net may *fire* if it is *enabled*, *i.e.* there are sufficient tokens in all of its input places; when the transition fires, it consumes the required input tokens, and creates tokens in its output places. A firing is atomic, i.e., a single non-interruptible step.

Unless an *execution policy* is defined, the execution of Petri nets is nondeterministic: when multiple transitions are enabled at the same time, any one of them may fire.

Since firing is nondeterministic, and multiple tokens may be present anywhere in the net (even in the same place), Petri nets are well suited for modeling the concurrent behavior of distributed systems.

Embedded Control Systems Design/Finite State Machines and Petri Nets

Finite State Machines (FSM) and Petri Nets (PN) are conceptual models to represent the discrete interactions in a system.

- A FSM is a conceptual model that represents how *one single activity* can change its behaviour over time, reaction to internally or externally triggered events.
- A PN is a conceptual representation of how *multiple activities* are coordinated.

FSM's and Petri Nets are at a higher level than the actual software. This means it explains the behavior, but not how it can be implemented. It is an ideal representation, it doesn’t take time delays in account.

Categories of system complexity

Systems of different complexity are represented in different ways. An FSM or UML statechart (similar to FSM, but with more possibilities) can only represent a centralized system, so there is no coordination needed with other systems of its type. Examples of systems of this type are a soda machine, an elevator, ...

Systems with distributed hardware and a centralized software state can be represented using Petri Nets. The centralized software coordinates how the different pieces of hardware act on each other.

A system with distributed hardware and distributed software state can also be represented by a Petri Net, but in this case there is no central control unit. The different pieces of hardware have their own software state and they all 'know' the Petri Net. Now the different pieces have to coordinate themselves.

An example of a distributed system is Robocup. This is a soccer game played by robots. Every robot is a system. If there is a control tower coordinating the robots, this is a case of a centralized software state. The control tower itself can be seen as a system of its own, represented by a statechart with states like 'ball controlled by own team', 'ball controlled by other team', ... In each state, the robots have to cooperate in a different way; this is represented by different Petri Nets.

If there is no control tower, each robot has to control itself and they have to coordinate themselves. Each robot then 'knows' the Petri Net and has to make its own decisions, which can introduce some problems. These are handled later in the section 'Petri Net'.

Petri Net

Petri Net(PN) is an abstract model to show the interaction between asynchronous processes. It is only one of the many ways to represent these interaction. Asynchronous means that the designer doesn't know when the processes start and in which sequence they'll take place. A common manner to visualize the concepts is with the use of places, tokens, transitions and arcs. We refer to the basics of Petri Net for a first introduction in notations. We want to mention that a transition can only fire when there are tokens in every input-place. When it fires, one token is taken from every input-place and every output-place from the transition gets an (extra) token.

File:Petrinet.png

Places can play the following roles:

- a type of communication medium: the wireless connection in Robocup;
- a buffer: all the robots send messages to each other. When one robot checks the first message, the other incoming information is placed in his memory;
- a geographical location: the environment of a robot, a place he has to go to;
- a possible state or state condition: normal mode and safe mode of a machine

Tokens can play the following roles:

- a physical object: a robot;

- an information object: a message between two robots;
- a collection of objects: the people mover;
- an indicator of a state: the state in which a robot is: defender/attacker;
- an indicator of a condition: a token indicates whether a certain condition is fulfilled (ex. Soccer game starts when the referee gives the signal).

Transitions can play the following roles:

- an event: start a thread, the *switching* of a machine from normal to safe mode;
- a transformation of an object: a robot who *changes* his role, see further;
- a transport of an object: the ball is *passed* between the robots.

An arc connects only places and transitions and indicates the direction in which the token travels.

RGPV PAPER QUESTIONS
Q.1 Explain important functions of Data Link Layer ? [RGPV June 2014]
Q.2 Explain Stop and wait Protocol ? [RGPV June 2014]
Q.3 Explain Sliding window Protocol ? [RGPV June 2014] [RGPV June 2013], [RGPV June 2012]
Q.4 What is Piggybacking ? What is HDLC [RGPV June 2014], [RGPV June 2012]
Q.5 Explain Selective repeat? [RGPV June 2013]
Q.6 Explain Finite state machine? [RGPV June 2013]
Q.7 Explain Bit Stuffing? [RGPV June 2013]