DATA COMMUNICATION AND COMPUTER NETWORKS

UNIT 3
Packet Switching Networks

Communication Networks: "Sets of nodes that are interconnected to allow the exchange of information such as voice, sound, graphics, pictures, video, text, data, etc…"

Telephone Networks: "The first well established and most widely used communication networks which are used for voice transmission"– Telephone networks originally used analog transmission as a transmission technology for the information. However, digital transmission started to evolve replacing a lot of the analog transmission techniques used in telephone networks.

Computer Networks: "Collection of autonomous computers interconnected by a technology to allow exchange of information"

A network is a series of connected devices. Whenever we have many devices, the interconnection between them becomes more difficult as the number of devices increases. Some of the conventional ways of interconnecting devices are

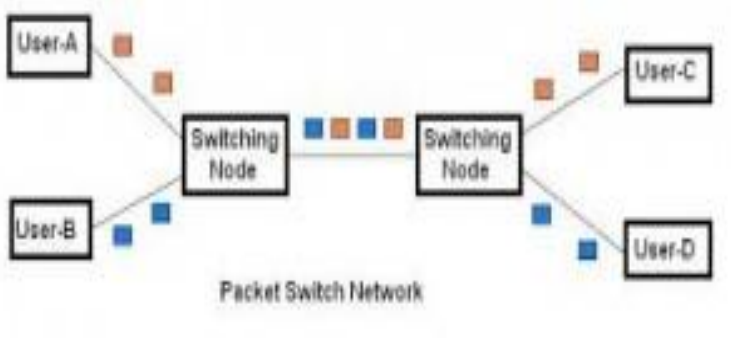
a. Point to point connection between devices as in mesh topology.
b. Connection between central device and every other device – as in star topology
c. Bus topology-not practical if the devices are at greater distances.

The solution to this interconnectivity problem is switching. A switched network consists of a series of interlinked nodes called switches. A switch is a device that creates temporary connections between two or more systems. Some of the switches are connected to end systems (computers and telephones) and others are used only for routing.

Packet Switching Network

In Packet Switching, messages are broken up into packets and each of which includes a header with source, destination and intermediate node address information. Individual Packets in packet switching technique take different routes to reach their respective destination. Independent routing of packets is done in this case for following reasons:

1. Bandwidth is reduces by the splitting of data onto different routes for a busy circuit.
2. For a certain link in the network, the link goes down during transmission the remaining packet can be sent through another route



Advantages of Packet switching

- The major advantage of Packet switching is that they are used for performing data rate conversion.
- When traversing the network switches, routers or the other network nodes then the packets are buffered in the queue, resulting in variable delay and throughput depending on the network's capacity and the traffic load on network.
- In cases where traffic fees are charged, for example in cellular communication, packet switching is characterized by a fee per unit of information transmitted.
- Packet switched networks are the building blocks of computer communication systems in which data units known as packets flow across the networks.
- It provides flexible communication in handling all kinds of connections for a wide range of applications e.g. telephone calls, video conferencing, distributed data processing etc...
- Packet switched networks with a unified, integrated data infrastructure known as the Internet can provide a variety of communication services requiring different bandwidths.
- To make efficient use of available resources, packet switched networks dynamically allocate resources only when required.
- The form of information in packet switched networks is always digital bits.

Network services and internal network operation
Essential function of network:

• The essential function of network is to transfer information among the users that are attached to the network.

• Transfer of information may be single block of information or sequence of blocks as shown in below figure.

• In case of single block of information, we are interested in having the block delivered correctly to destination and also interested in delay experienced in traversing the network.

• In case of sequence of blocks, we are interested not only in receiving the blocks correctly and in right sequence.

Network service

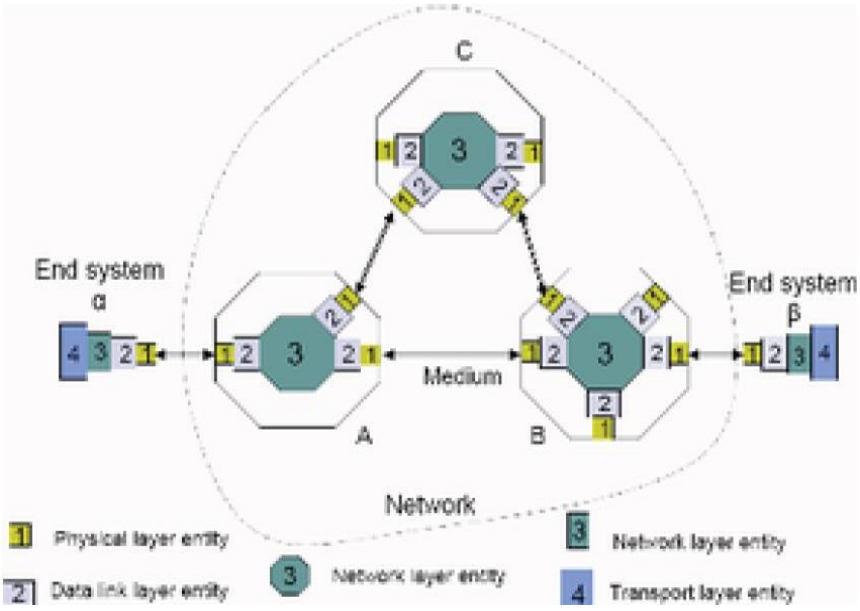
Network service can be Connection-oriented service or connectionless service Connectionless service:

Connectionless service is simple with two basic interactions (1) a request to network layer that it send a packet (2) an indication from the network layer that a packet has arrived.. It puts total responsibility of error control, sequencing and flow control on the end system transport layer

Connection-oriented service

- The Transport layer cannot request transmission of information until a connection is established between end systems
- Network layer must be informed about the new flow
- Network layer maintains state information about the flows it is handling
- During connection set up, parameters related to usage and quality of services may be negotiated and network resources may be allocated
- Connection release procedure may be required to terminate the connection
- It is also possible for a network layer to provide a choice of services to the user of network like:
- best-effort connectionless services
- Low delay connectionless services
- Connection oriented reliable stream services
- Connection oriented transfer of packets with guaranteed delay and bandwidth

Packet network can be viewed from two perspectives
External view of network: It is concerned with the services that the network provides to the transport layer
Internal operation of the network:

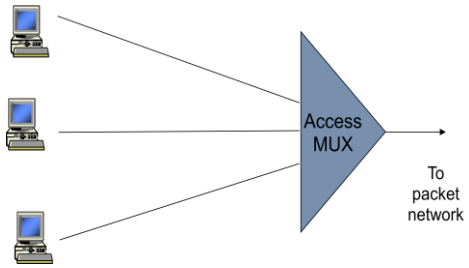


The fig above shows the relation between the service offered by the network and the internal network operation

- The internal operation of the network is connectionless if packets are transferred within the network as datagrams.
- Each packets are routed independently.
- Packets follow different paths from end to end and arrive out of order.
- The internal operation of the network is connection-oriented if packets follow a virtual circuit along a path that has been established from source to destination.
- Virtual circuit setup is done once, then packets are simply forwarded.
- If resources are reserved then bandwidth, delay and loss guarantees are provided.

Packet Network Topology
Ex1: Access Multiplexing

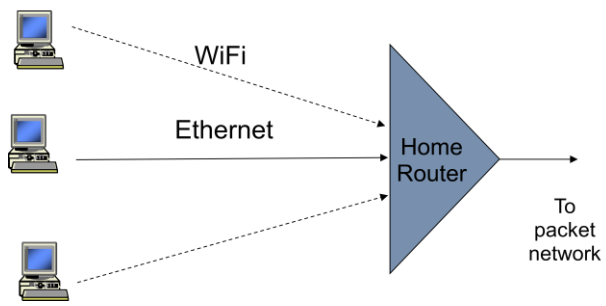
It is the way in which users access packet networks. Figure shows an access multiplexer where the packets from a number of users share a transmission line.
The multiplexer combines the typically bursty rows of the individual computers into aggregated rows that make efficient use of the transmission line. Note that different applications within a single computer can generate multiple simultaneous rows to different destinations.



The diagram above shows an access network with a point to point topology where computer in homes are connected to an access multiplexer located in service provider network. The main purpose of the access multiplexer is to combine the bursty traffic flows from individual computers into aggregated flows.

- DSL traffic multiplexed at DSL Access Mux
- Cable modem traffic multiplexed at Cable Modem Termination System
- Private IP addresses in Home is done using Network Address Translation (NAT).

EX2:Home LANS

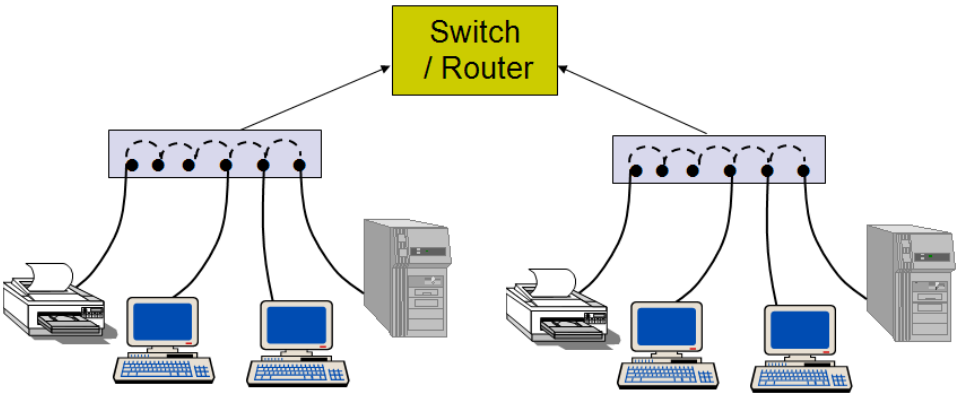


Home Router

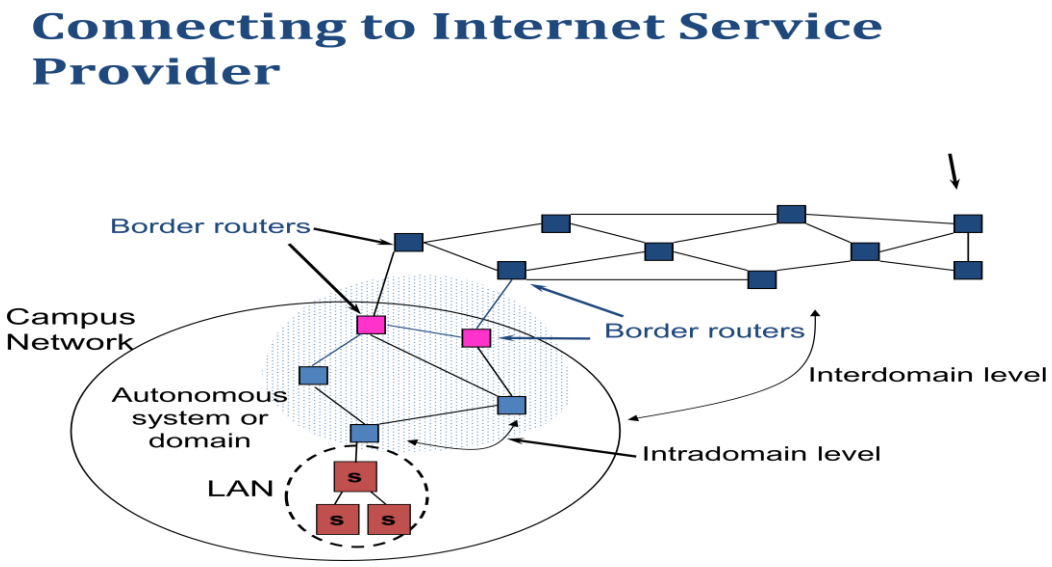
- LAN Access using Ethernet or WiFi (IEEE 802.11)
- Private IP addresses in Home (192.168.0.x) using Network Address Translation (NAT)
- Single global IP address from ISP issued using Dynamic Host Configuration Protocol (DHCP)

LAN Concentration

LAN Hubs and switches in the access network also aggregate packet streams that flows into switches and routers.



Ex3: Connecting to Internet Service Provide



Domain: the routers running the same routing protocol

- Autonomous System: one or more domains under the single administration.
- The campus network maybe connected to internet service provider (ISP) through one or more border routers.
- To communicate with other networks, the autonomous system must provide information about its network routes in border routers.

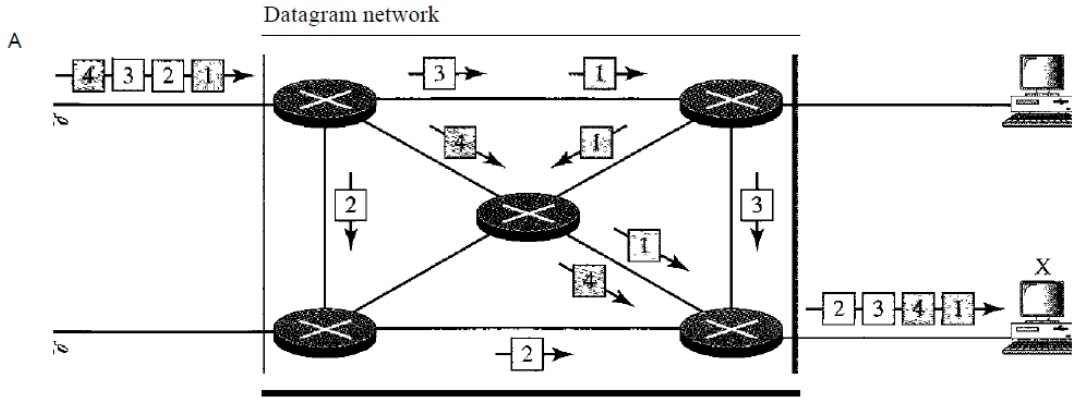
- The border router communicates on an interdomain level, whereas other routers operate at an intradomain level.

Datagram network

In data communications, we need to send messages from one end system to another. If the message is going to pass through a packet-switched network, it needs to be divided into packets of fixed or variable size. The size of the packet is determined by the network and the governing protocol.

In packet switching, there is no resource allocation for a packet. This means that there is no reserved bandwidth on the links, and there is no scheduled processing time for each packet. Resources are allocated on demand. The allocation is done on a first come, first-served basis. When a switch receives a packet, no matter what is the source or destination, the packet must wait if there are other packets being processed. As with other systems in our daily life, this lack of reservation may create delay. For example, if we do not have a reservation at a restaurant, we might have to wait. we do not have a reservation at a restaurant, we might have to wait. In a packet-switched network, there is no resource reservation; resources are allocated on demand.

In a datagram network, each packet is treated independently of all others. Even if a packet is part of a multipacket transmission, the network treats it as though it existed alone. Packets in this approach are referred to as datagrams.
Datagram switching is normally done at the network layer. We briefly discuss datagram networks here as a comparison with circuit-switched and virtual-circuits witched networks.
Figure shows how the datagram approach is used to deliver four packets from station A to station X. The switches in a datagram network are traditionally referred to as routers.

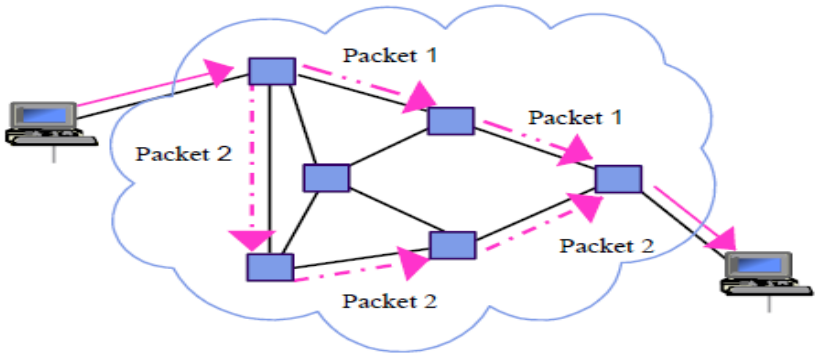


Virtual circuits

Virtual circuit switching is a packet switching methodology whereby a path is established between the source and the final destination through which all the packets will be routed during a call. This path is called a virtual circuit because to the user, the connection appears to be a dedicated physical circuit. However, other communications may also be sharing the parts of the same path.

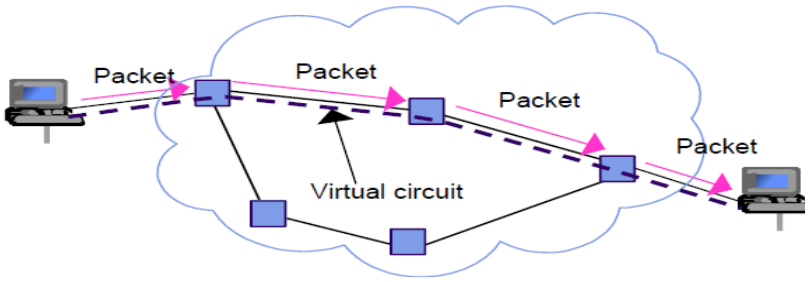
Before the data transfer begins, the source and destination identify a suitable path for the virtual circuit. All intermediate nodes between the two points put an entry of the routing in their routing table for the call. Additional parameters, such as the maximum packet size, are also exchanged between the source and the destination during call setup. The virtual circuit is cleared after the data transfer is completed.

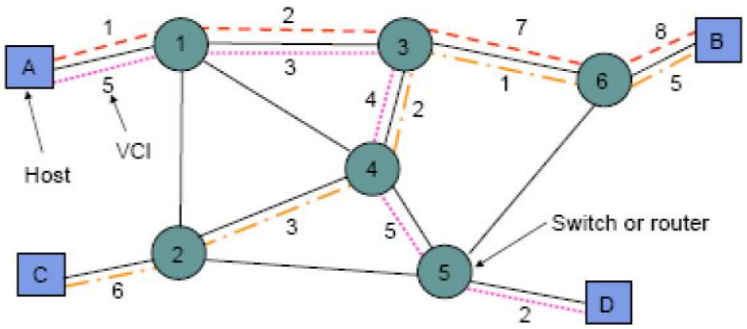
Datagram or Connectionless Packet Switching



- Messages broken into smaller units (packets)
- Source & destination addresses in packet header
- Connectionless, packets routed independently (datagram)
- When a message arrives at the packet switch, the destination address is Examined to determine the next hop.
- Packet may arrive out of order
- Re-sequencing maybe required at destination.
- Pipelining of packets across network can reduce delay, increase throughput
- Lower delay that message switching, suitable for interactive traffic

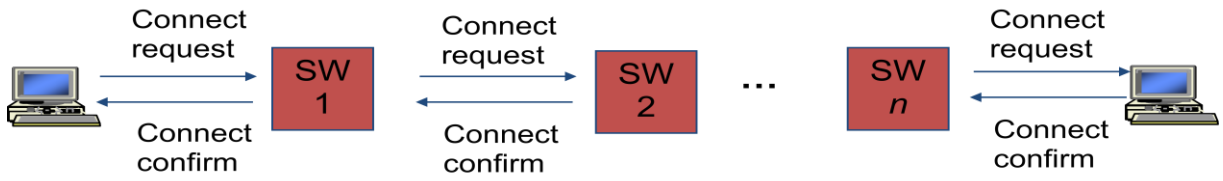
Virtual – Circuit Packet Switching





- Before the transmission of packets, it involves establishment of virtual circuit between source and destination
- All packets follow the same path
- Abbreviated header identifies connection on each link
- Packets queue for transmission
- Variable bit rates possible

Connection setup



- Signaling messages propagate as route is selected
- Signaling messages identify connection and setup tables in switches
- Typically a connection is identified by a local tag, Virtual Circuit Identifier (VCI)
- Each switch only needs to know how to relate an incoming tag in one input to an outgoing tag in the corresponding output.
- Once tables are setup, packets can flow along path

Connection Setup Delay

- Call set-up phase sets ups pointers in fixed path along network
- All packets for a connection follow the same path
- Abbreviated header identifies connection on each link
- Packets queue for transmission
- Variable bit rates possible, negotiated during call set-up
- Delays variable, cannot be less than circuit switching

Routing

Routing is the process of selecting a path for traffic in a network, or between or across multiple networks. Routing is performed for many types of networks, including circuit-switched networks, such as the public switched telephone network (PSTN), computer networks, such as the Internet.

In internetworking, the process of moving a packet of data from source to destination. Routing is usually performed by a dedicated device called a router. Routing is a key feature of the Internet because it enables messages to pass from one computer to another and eventually reach the target machine. Each intermediary computer performs routing by passing along the message to the next computer. Part of this process involves analyzing a routing table to determine the best path.

Routing: it is concerned with determining feasible path for packets to follow from each source to destination. Which path is “best”? The term best depends on the objective function that network operator tries to optimize which maybe Min delay, Min hop, Max bandwidth, Min cost, Max reliability

Routing Table

A routing table contains the information necessary to forward a packet along the best path toward its destination. Each packet contains information about its origin and destination. When a packet is received, a network device examines the packet and matches it to the routing table entry providing the best match for its destination. The table then provides the device with instructions for sending the packet to the next hop on its route across the network. A basic routing table includes the following information.

- Destination: The IP address of the packet's final destination

- Next hop: The IP address to which the packet is forwarded
- Interface: The outgoing network interface the device should use when forwarding the packet to the next hop or final destination
- Metric: Assigns a cost to each available route so that the most cost-effective path can be chosen
- Routes: Includes directly-attached subnets, indirect subnets that are not attached to the device but can be accessed through one or more hops, and default routes to use for certain types of traffic or when information is lacking.

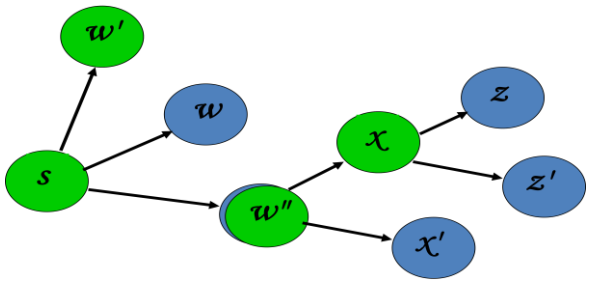
Tables for static network devices do not change unless a network administrator manually changes them. In dynamic routing, devices build and maintain their routing tables automatically by using routing protocols to exchange information about the surrounding network topology.

Dynamic routing tables allow devices to "listen" to the network and respond to occurrences like device failures and network congestion. Flooding is a simple routing technique in computer networks where a source or node sends packets through every outgoing link.

Dijkstra Algorithm: Finding shortest paths in order

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was **conceived** by computer scientist Edsger W. Dijkstra in 1956 and published three years later.

Find shortest paths from source s to all other destinations



Closest node to s is 1 hop

2nd closest node to s is 1 hop away from s or w"

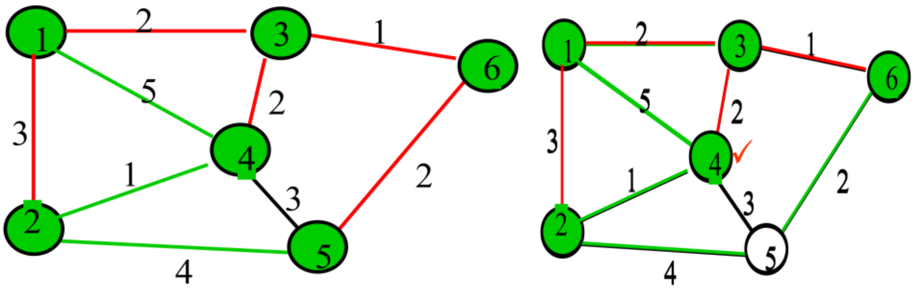
3rd closest node to s is 1 hop away from s, w", or x

Dijkstra's algorithm

- N: set of nodes for which shortest path already found
- Initialization: (S*tart with source node s)*
  - $N = \{s\}$, $D_s = 0$, "s is distance zero from itself"
  - $D_j = C_{sj}$ for all $j \neq s$, distances of directly-connected neighbors
- Step A: (*Find next closest node i*)
  - Find $i \notin N$ such that
  - D_i = min Dj for $j \notin N$
  - Add i to N
  - If N contains all the nodes, stop
- Step B: (*update minimum costs)*
  - For each node $j \notin N$
  - D_j = min (D_j, $D_i + C_{ij}$)
  - Go to Step A

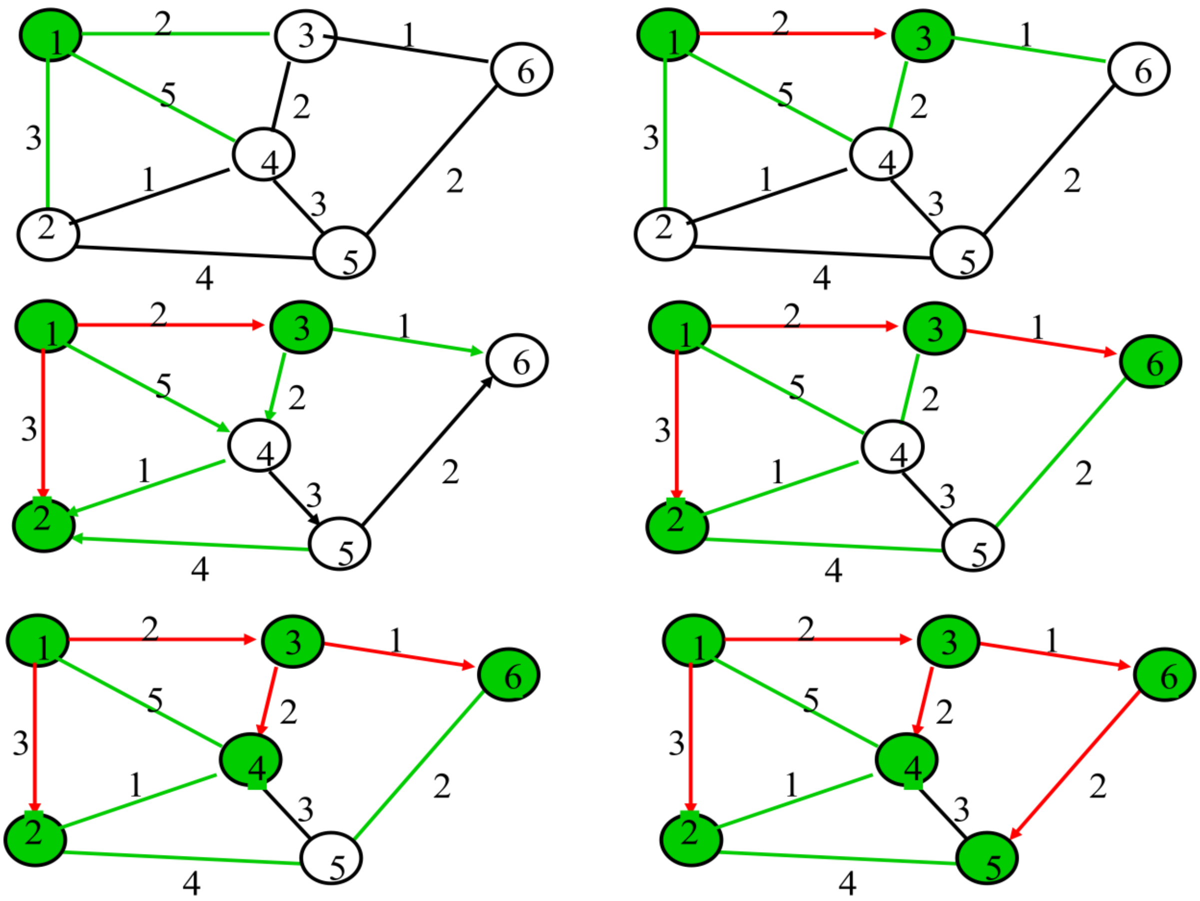
Minimum distance from s to *j through node* ***i*** *in N*

Execution of Dijkstra's algorithm



Iteration	N	D_2	D_3	D_4	D_5	D_6
Initial	{1}	3	2 ✓	5	∝	∝
1	{1,3}	3 ✓	2	4	∝	3
2	{1,2,3}	3	2	4	7	3 ✓
3	{1,2,3,6}	3	2	4 ✓	5	3
4	{1,2,3,4,6}	3	2	4	5 ✓	3
5	{1,2,3,4,5,6}	3	2	4	5	3

Shortest Paths in Dijkstra's Algorithm



Flooding

In a network, flooding is the forwarding by a router of a packet from any node to every other node attached to the router except the node from which the packet arrived. Flooding is a way to distribute routing information updates quickly to every node in a large network. It is also sometimes used in multicast packets (from one source node to many specific nodes in a real or virtual network.

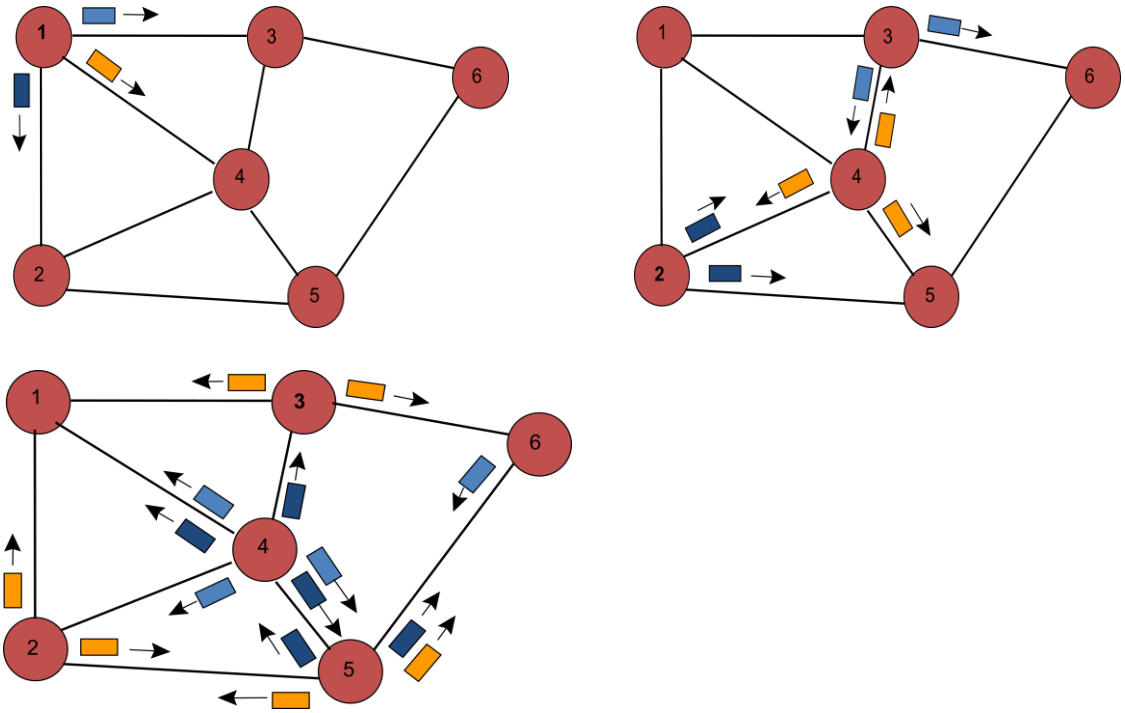
Principle of flooding: a node (or a packet switch) forwards an incoming packet to all ports except to the one it arrived on. Each node (a switch) performs the flooding process such that the packet will reach the destination as long as at least one path exists between the source and the destination.

Sends a packet to all nodes in a network

- No routing tables available
- Need to broadcast packet to all nodes (e.g. to propagate link state information)

Approach

- Send packet on all ports except one from where it arrived
- Exponential growth in packet transmissions
- Flooding generates vast numbers of duplicate packets



Limited Flooding

- Time-to-Live field in each packet limits number of hops to certain diameter
- Each switch adds its ID before flooding; discards repeats
- Source puts sequence number in each packet; switches records source address and sequence number and discards repeats

There are three methods to reduce the resource consumption in the network
1) Use a time-to-live (TTL) field in each packet.

- When the source sends a packet, the time-to-live field is initially set to some small number.

- Each node decrements the field by one before flooding the packet. If the value reaches zero, the switch discards the packet.
- To avoid unnecessary waste of bandwidth, the time-to-live should ideally be set to the minimum hop number between two furthest nodes (called the diameter of the network).

2) Add an identifier before flooding

- Every node adds an identifier before flooding
- When a node identifies a packet that contains the identifier of the switch, it discards the packet.
- This method effectively prevents a packet from going around a loop.

3) Have a unique sequence number

- Each packet from the given source is uniquely identified with a sequence number
- When a node receives a packet, it records the source address and the sequence number
- If node discovers that packet has already visited the node, it will discard the packet

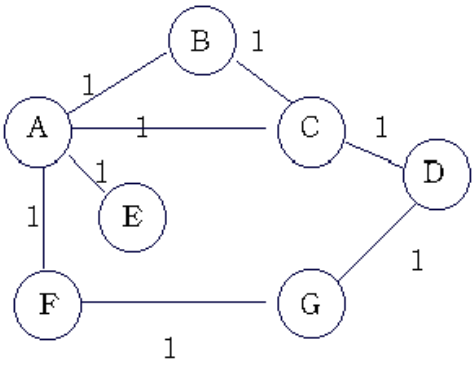
Distance-Vector Routing

Each node constructs a one-dimensional array containing the "distances"(costs) to all other nodes and distributes that vector to its immediate neighbors.

1. The starting assumption for distance-vector routing is that each node knows the cost of the link to each of its directly connected neighbors.

2. A link that is down is assigned an infinite cost.

Example



Information	**Distance to Reach Node**						
Stored at Node	**A**	**B**	**C**	**D**	**E**	**F**	**G**
A	0	1	1	∞	1	1	∞
B	1	0	1	∞	∞	∞	∞
C	1	1	0	1	∞	∞	∞
D	∞	∞	1	0	∞	∞	1

E	1	□	□	□	0	□	□
F	1	□	□	□	□	0	1
G	□	□	□	1	□	1	0

We can represent each node's knowledge about the distances to all other nodes as a table like the one given in Table 1.Note that each node only knows the information in one row of the table.

1. Every node sends a message to its directly connected neighbors containing its personal list of distance. (for example, **A** sends its information to its neighbors **B,C,E**, and **F**.)

2. If any of the recipients of the information from **A** find that **A** is advertising a path shorter than the one they currently know about, they update their list to give the new path length and note that they should send packets for that destination through **A**. (node **B** learns from **A** that node **E** can be reached at a cost of 1; **B** also knows it can reach **A** at a cost of 1, so it adds these to get the cost of reaching **E** by means of **A**. **B** records that it can reach **E** at a cost of 2 by going through **A**.)

3. After every node has exchanged a few updates with it's directly connected neighbors, all nodes will know the least-cost path to all the other nodes.

4. In addition to updating their list of distances when they receive updates, the nodes need to keep track of which node told them about the path that they used to calculate the cost, so that they can create their forwarding table. (for example, **B** knows that it was **A** who said " I can reach **E** in one hop" and so **B** puts an entry in its table that says " To reach **E**, use the link to **A**.)

Information	**Distance to Reach Node**						
Stored at Node	**A**	**B**	**C**	**D**	**E**	**F**	**G**
A	0	1	1	2	1	1	2
B	1	0	1	2	2	2	3
C	1	1	0	1	2	2	2
D	□	2	1	0	3	2	1
E	1	2	2	3	0	2	3
F	1	2	2	2	2	0	1
G	□	3	2	1	3	1	0

Final distances stored at each node (global view).

Congestion

A state occurring in network layer when the message traffic is so heavy that it slows down network response time. Effects of Congestion

As delay increases, performance decreases.

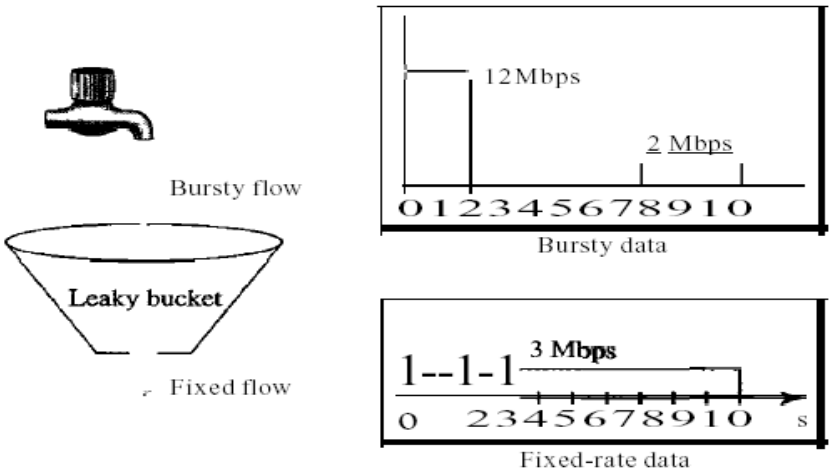
If delay increases, retransmission occurs, making situation worse.

Congestion control algorithms: Leaky Bucket Algorithm

Let us consider an example to understand. Imagine a bucket with a small hole in the bottom. No matter at what rate water enters the bucket, the outflow is at constant rate. When the bucket is full with water additional water entering spills over the sides and is lost.

Leaky Bucket

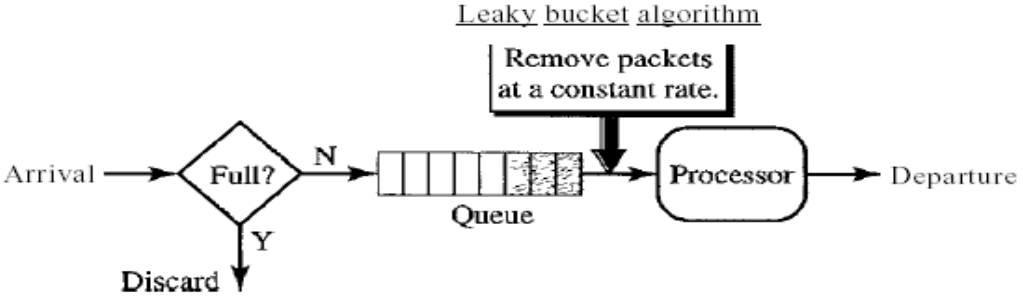
If a bucket has a small hole at the bottom, the water leaks from the bucket at a constant rate on the rate at which the water is input to the bucket unless the bucket is empty. The input rate can vary, but the output rate remains constant. Similarly, in networking, a technique called leaky bucket can smooth out bursty traffic. Bursty chunks are stored in the bucket and sent out at an average rate. Figure shows a leaky bucket and its effects.



In the figure, we assume that the network has committed a bandwidth of 3 Mbps for a host. The use of the leaky bucket shapes the input traffic to make it conform to this commitment. The Figure shows that the host sends a burst of data at a rate of 12 Mbps for 2 s, for a total of 24 Mbits of data.

The host is silent for 5 s and then sends data at a rate of 2 Mbps for 3 s, for a total of 6 Mbits of data. In all, the host has sent 30 Mbits of data in lOs. The leaky bucket smooths the traffic by sending out data at a rate of 3 Mbps during the same 10 s. Without the leaky bucket, the beginning burst may have hurt the network by consuming more bandwidth than is set aside for this host. We can also see that the leaky bucket may prevent congestion. As an analogy, consider the freeway during rush hour (bursty traffic). If, instead, commuters could stagger their working hours, congestion on our freeways could be avoided.

A simple leaky bucket implementation is shown in the Figure below. A FIFO queue holds the packets. If the traffic consists of fixed-size packets



The process removes a fixed number of packets from the queue at each tick of the clock. If the traffic consists of variable-length packets, the fixed output rate must be based on the number of bytes or bits. The following is an algorithm for variable-length packets:

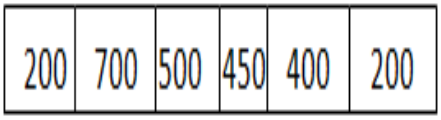
1. Initialize a counter to n at the tick of the clock.

2. If n is greater than the size of the packet, send the packet and decrement the counter by the packet size. Repeat this step until n is smaller than the packet size.

3. Reset the counter and go to step 1.

A leaky bucket algorithm shapes bursty traffic into fixed-rate traffic by averaging the data rate. It may drop the packets if the bucket is full.

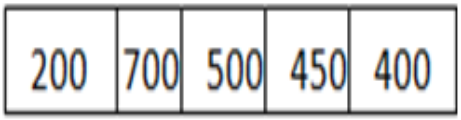
Example: Let n=1000



Packet=200 since n> front of Queue i.e. n>200

Therefore, n=1000-200=800

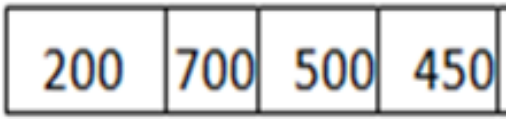
Packet size of 200 is sent to the network.



Now Again n>front of the queue i.e. n > 400

Therefore, n=800-400=400

Packet size of 400 is sent to the network.



Since n< front of queue Therefore, the procedure is stop.

And we initialize n=1000 on another tick of clock.

This procedure is repeated until all the packets are sent to the network.

Data Link Issues

As we know that computer network means connecting two or more node with each other and establishing communication between them. When we talk about communication in computer network we come across various transmission impairments which can be caused by

1. Attenuation
2. Distortion and
3. Noise

Error Detection and Correction

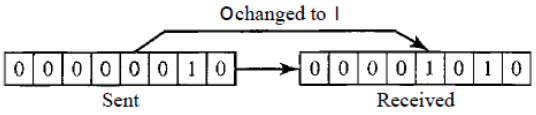
Networks must be able to transfer data from one device to another with acceptable accuracy. For most applications, a system must guarantee that the data received are identical to the data transmitted. Any time data are transmitted from one node to the next, they can become corrupted in passage. Many factors can alter one or more bits of a message. Data can be corrupted during transmission. Some applications require a mechanism for detecting and correcting errors. Some applications can tolerate a small level of error. For example, random errors in audio or video transmissions may be tolerable, but when we transfer text, we expect a very high level of accuracy.

Types of Errors

Whenever bits flow from one point to another, they are subject to unpredictable changes because of interference. This interference can change the shape of the signal. In a single-bit error, a 0 is changed to a 1 or a 1 to a 0. In a burst error, multiple bits are changed. For example, a 11100 s burst of impulse noise on a transmission with a data rate of 1200 bps might change all or some of the12 bits of information.

1. Single-Bit Error

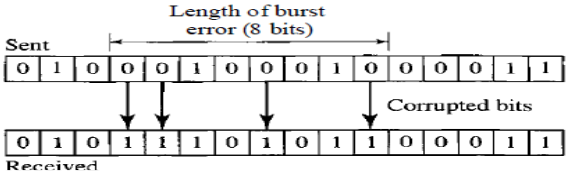
The term *single-bit error* means that only 1 bit of a given data unit (such as a byte, character, or packet) is changed from 1 to 0 or from 0 to 1. In a single-bit error, only 1 bit in the data unit has changed. Figure shows the effect of a single-bit error on a data unit. To understand the impact of the change, imagine that each group of 8 bits is an ASCII character with a 0 bit added to the left. In Figure 10.1,00000010 (ASCII *STX)* was sent, meaning *start of text,* but 00001010 (ASCII *LF)* was received, meaning *line feed.* (For more information about ASCII code, see Appendix A.)



Single-bit errors are the least likely type of error in serial data transmission. To understand why, imagine data sent at 1 Mbps. This means that each bit lasts only 1/1,000,000 s, or 1)ls. For a single-bit error to occur, the noise must have a duration of only 1)ls, which is very rare; noise normally lasts much longer than this.

2. Burst Error

The term burst error means that 2 or more bits in the data unit have changed from 1 to 0 or from 0 to 1. A burst error means that 2 or more bits in the data unit have changed. Figure 10.2 shows the effect of a burst error on a data unit. In this case, 0100010001000011 was sent, but 0101110101100011 was received. Note that a burst error does not necessarily mean that the errors occur in consecutive bits. The length of the burst is measured from the first corrupted bit to the last corrupted bit. Some bits in between may not have been corrupted.



A burst error is more likely to occur than a single-bit error. The duration of noise is normally longer than the duration of 1 bit, which means that when noise affects data, it affects a set of bits. The number of bits affected depends on the data rate and duration of noise. For example, if we are sending data at I kbps, a noise of 11100 s can affect 10 bits; if we are sending data at I Mbps, the same noise can affect 10,000 bits.

Redundancy: The central concept in detecting or correcting errors is redundancy. To be able to detect or correct errors, we need to send some extra bits with our data. These redundant bits are added by the sender and removed by the receiver. Their presence allows the receiver to detect or correct corrupted bits.

To detect or correct errors, we need to send extra (redundant) bits with data.

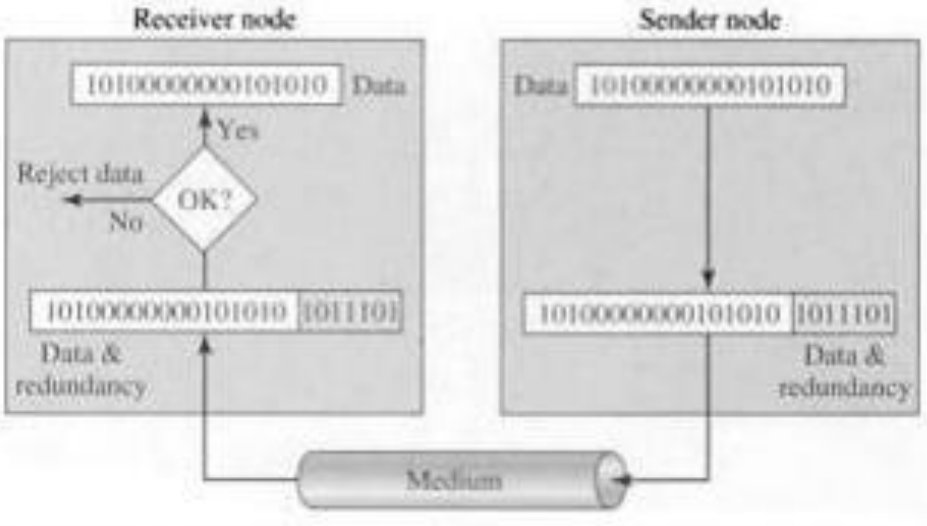
Error Detection

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.

Redundancy

The central concept in detecting or correcting errors is redundancy. To be able to detect or correct errors, we need to send some extra bits with our data. These redundant bits are added by the sender and removed by the receiver. Their presence allows the receiver to detect or correct corrupted bits. The concept of including extra information in the transmission for error detection is a good one. But instead of repeating the entire data stream, a shorter group of bits may be appended to the end of each unit. This technique is called redundancy because the extra bits are redundant to the information: they are discarded as soon as the accuracy of the transmission has been determined.

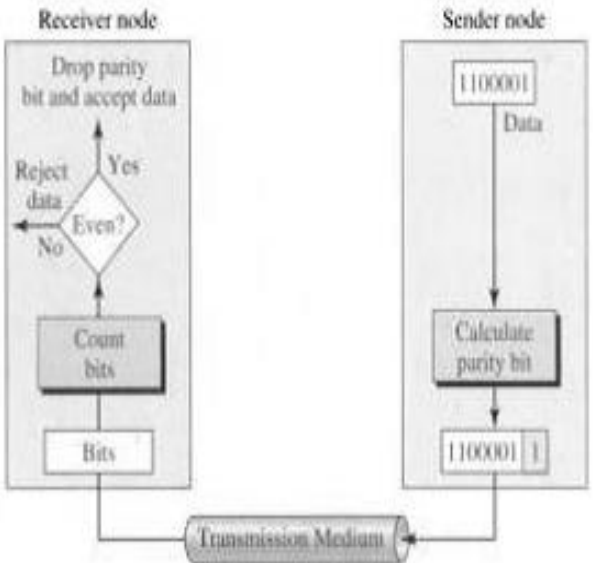
Figure shows the process of using redundant bits to check the accuracy of a data unit. Once the data stream has been generated, it passes through a device that analyses it and adds on an appropriately coded redundancy check. The data unit, now enlarged by several hits, travels over the link to the receiver. The receiver puts the entire stream through a checking function. If the received hit stream passes the checking criteria, the data portion of the data unit is accepted and the redundant bits are discarded.



Three types of redundancy checks are common in data communications: parity check, cyclic redundancy check (CRC) and checksum

Simple Parity Check:

In this technique, a redundant bit called a parity bit is added to every data unit so that the total number of 1's in the unit (including the parity bit) becomes even (or odd). Figure (10) shows this concept when transmit the binary data unit 1100001.



Performance of Simple Parity Check:

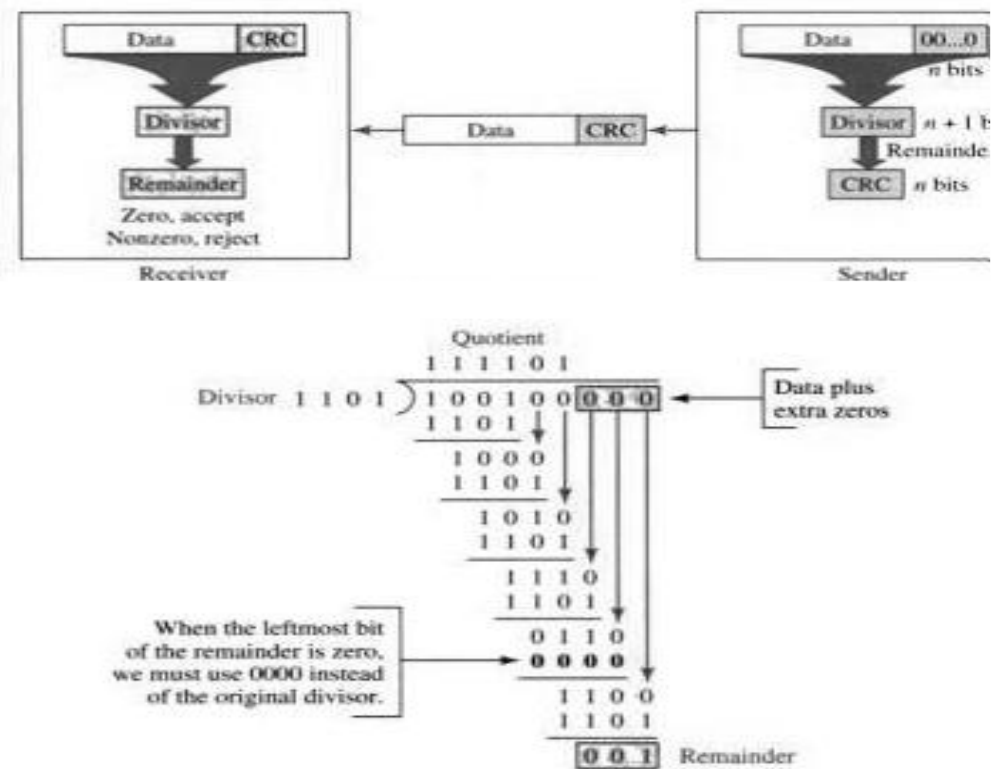
Simple parity check can detect all single-bit errors. It can also detect burst errors as long as the total number of bits changed is odd. This method cannot detect errors where the total number of hits changed is even. If any two bits change in transmission, the changes cancel each other and the data unit will pass a parity check even though the data unit is damaged. The same holds true for any even lumber of errors.

Cyclic Redundancy Check (CRC)

The third and most powerful of the redundancy checking techniques is the cyclic redundancy check (CRC). Unlike the parity check which is based on addition. CRC is based on binary division. In CRC, instead of adding bits to achieve a desired parity, a sequence of redundant bits, called the CRC or the CRC remainder, is appended to the end of a data unit so that the resulting data unit becomes exactly divisible by a second, predetermined binary number. At its destination, the incoming data unit is divided by the same number. If at this step there is no remainder the data unit is assumed to be intact and is therefore accepted. A remainder indicates that the data unit has been damaged in transit and therefore must be rejected.

The redundancy bits used by CRC are derived by dividing the data unit by a predetermined divisor; the remainder is the CRC. To be valid, a CRC must have two qualities:

1. It must have exactly one less bit than the divisor, and appending it to the end of the data string must make the resulting bit sequence exactly divisible by the divisor.
2. Both the theory and the application of CRC error detection are straightforward. The only complexity is in deriving the CRC. To clarify this process, we will start with an overview and add complexity as we go. Figure 12 provides an outline of the basic steps.



Performance of CRC:

CRC is a very effective error detection method. If the divisor is chosen according to the previously mentioned rules,

1. CRC can detect all burst errors that affect an odd number of bits.
2. CRC can detect all burst errors of length less than or equal to the degree of the polynomial
3. CRC can detect, with a very high probability, burst errors of length greater than the degree of the polynomial.

Checksum

The error detection method used by the higher-layer protocols is called checksum. Checksum is based on the concept of redundancy.

Checksum Generator: In the sender, the checksum generator subdivides the data unit into equal segment of n bit (usually 16). These segments are added together using one's complement arithmetic in such a way that the total is also n bit long. That total is then complemented and appended to the end of the original data unit as redundancy bits, called the checksum field. The extended data unit is transmitted across the network. So if the sum of the data segment is T, the checksum will be – T.

Checksum Checker: The receiver subdivides the data unit as a above and adds all segments together and complements the result. If the extended data unit is intact, the total value found by adding the data segments and the checksum field should be zero. If the result is not zero, the packet contains an error and the rejects it.

The sender follows these steps:

1. the unit is divided into k sections, each n bits
2. All sections are added together using one's complement to get the sum.

3. The sum is complemented and becomes the checksum.
4. The checksum is sent with the data

The receiver follows these steps:

1. The unit is divided into K sections, each of n bits.
2. All sections are added together using one's complement to get the sum.
3. The sum is complemented
4. If the result is zero, the data are accepted; otherwise, they are rejected.

Ex: 1. suppose the following block of 16 bits is to be sent using checksum of 8 bits.

10101001 00111001
The numbers are added using one's complement arithmetic
10101001
00111001
11100010 sum
00011101 Checksum
The pattern sent is:
10101001 00111001 00011101

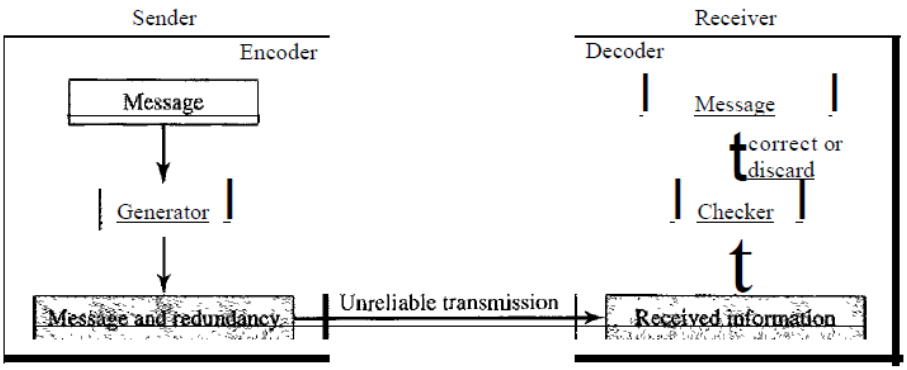
Detection versus Correction

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

In error correction, we need to know the exact number of bits that are corrupted and more importantly, their location in the message. The number of the errors and the size of the message are important factors. If we need to correct one single error in an 8-bit data unit, we need to consider eight possible error locations; if we need to correct two errors in a data unit of the same size, we need to consider 28 possibilities. You can imagine the receiver's difficulty in finding 10 errors in a data unit of 1000 bits.

Coding

Redundancy is achieved through various coding schemes. The sender adds redundant bits through a process that creates a relationship between the redundant bits and the actual data bits. The receiver checks the relationships between the two sets of bits to detect or correct the errors. The ratio of redundant bits to the data bits and the robustness of the process are important factors in any coding scheme. Figure shows the general idea of coding.



The sender creates codewords out of datawords by using a generator that applies the rules and procedures of encoding (discussed later). Each codeword sent to the receiver may change during transmission. If the received codeword is the same as one of the valid codewords, the word is accepted; the corresponding dataword is extracted for use. If the received codeword is not valid, it is discarded. However, if the codeword is corrupted during transmission but the received word still matches a valid codeword, the error remains undetected.This type of coding can detect only single errors. Two or more errors may remain undetected.

Example

Let us assume that k =2 and n =3. Table shows the list of datawords and codewords. Later, we will see how to derive a codeword from a dataword.

Table: A code for error detection

Datawords	Codewords
00	000
01	011
10	101
11	110

Assume the sender encodes the dataword 01 as 011 and sends it to the receiver. Consider the following cases:

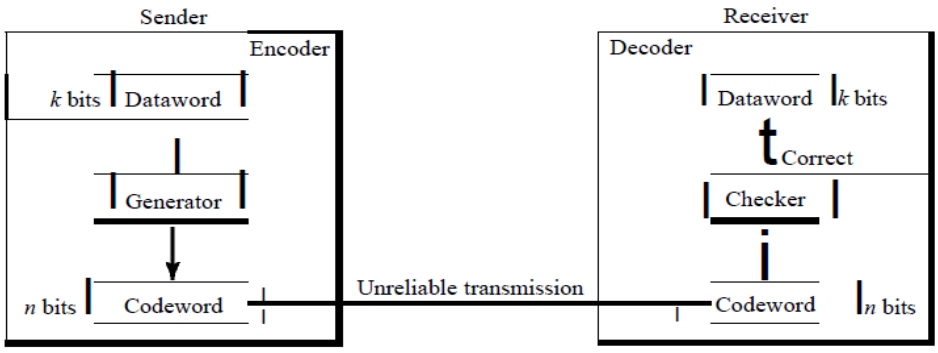
1. The receiver receives OIl. It is a valid codeword. The receiver extracts the dataword 01 from it.

2. The codeword is corrupted during transmission, and 111 is received (the leftmost bit is corrupted). This is not a valid codeword and is discarded.

3. The codeword is corrupted during transmission, and 000 is received (the right two bits are corrupted). This is a valid codeword. The receiver incorrectly extracts the dataword 00. Two corrupted bits have made the error undetectable.

Error Correction

As we said before, error correction is much more difficult than error detection. In error detection, the receiver needs to know only that the received codeword is invalid; in error correction the receiver needs to find (or guess) the original codeword sent. We can say that we need more redundant bits for error correction than for error detection. Figure shows the role of block coding in error correction. We can see that the idea is the same as error detection but the checker functions are much more complex.



Single bit Error Correction

The concept underlying error correction can be most easily understood by examining the simplest case: Single-bit errors As we saw earlier, single-bit errors can be detected by the addition of a redundant(parity) bit to the data unit (VRC). A single additional bit can detect single-bit errors in any sequence of bits because it must distinguish between only two conditions: error or no error. A bit has two state (0 and 1). These two states are sufficient for this level of detection.

But what if we want to correct as well as detect single-bit errors? Two state are enough to detect an error but not to correct it. An error occurs when the receiver reads a 1 bit as a 0 or a 0 bit as a 1. To correct the error, the receive simply decreases the value of the altered bit. To do so, however, it must know which bit is error. The secret of error correction, therefore, is to locate the invalid bit or bits.

EX: To correct a single-bit error in an ASCII character, the error correction code must determine which of the seven bits has changed. In this case, we have to distinguish between eight different states: no error, error in position 1, error in position2, and so on, up to error in position 7. to do so requires enough redundancy bits to show all eight states. In a single-bit error, only 1 bit in the data unit has changed

Hamming Distance

One of the central concepts in coding for error control is the idea of the Hamming distance. The Hamming distance between two words (of the same size) is the number of differences between the corresponding bits. We show the Hamming distance between two words *x* and *y* as *d(x, y).* The Hamming distance can easily be found if we apply the XOR operation on the two words and count the number of Is in the result. Note that the Hamming distance is a value greater than zero.

The Hamming distance between two words is the number of differences between corresponding bits.
Let us find the Hamming distance between two pairs of words.

1. The Hamming distance *d(OOO,* 011) is 2 because 000 $\oplus$ 011 is 011 (two Is).
2. The Hamming distance *d(10101,* 11110) is 3 because 10101 $\oplus$ 11110 is 01011 (three Is).

Minimum Hamming Distance

Although the concept of the Hamming distance is the central point in dealing with error detection and correction codes, the measurement that is used for designing a code is the minimum Hamming distance. In a set of words, the minimum Hamming distance is the smallest Hamming distance between all possible pairs. We use dmin to define the minimum Hamming distance in a coding scheme. To find this value, we find the Hamming distances between all words and select the smallest one.
The minimum Hamming distance is the smallest Hamming distance between all possible pairs in a set of words.

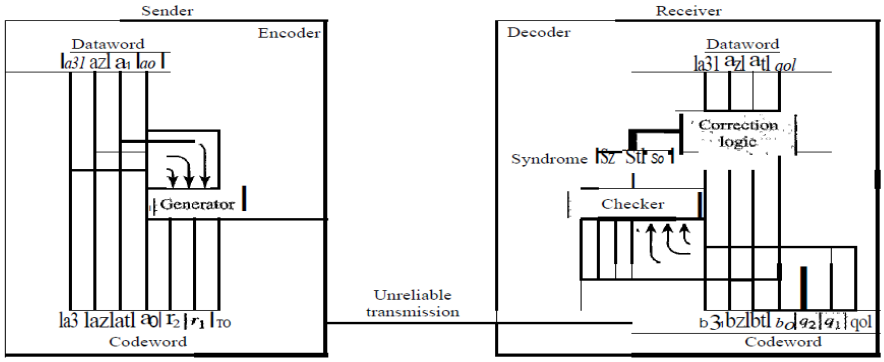
Linear Block Codes: These codes are trivial because we can easily find the encoding and decoding algorithms and check their performances.

Hamming Codes

Hamming code or Hamming Distance Code is the best error correcting code we use in most of the communication network and digital systems. Hamming code is a set of error-correction codes that can be used to **detect and correct the errors** that can occur when the data is moved or stored from the sender to

the receiver. These codes were originally designed with *dmin* = 3, which means that they can detect up to two errors or correct one single error. Although there are some Hamming codes that can correct more than one error, our discussion focuses on the single-bit error-correcting code. First let us find the relationship between *n* and *k* in a Hamming code. We need to choose an integer *m* >= 3. The values of nand *k* are then calculated from mas *n* = *2m* – 1 and *k* ::: *n* - *m*. The number of check bits *r* =*m*.
For example, if*m* =3, then *n* ::: 7 and *k:::* 4. This is a Hamming code *C(7,* 4) with dmin =3.
Table shows the datawords and codewords for this code.

Datawords	*Codewords*	*Datawords*	*Codewords*
0000	0000000	1000	1000110
0001	0001101	1001	1001011
0010	0010111	1010	1010001
0011	0011010	1011	10111 00
0100	0100011	1100	1100101
0101	01011 10	1101	1101000
0110	0110100	1110	1110010
0111	0111001	1111	1111111



A copy of a 4-bit dataword is fed into the generator that creates three parity checks
$r_0=a_2+a_1+a_0$, $r_1=a_3+a_2+a_1$ and $r_2=a_1+a_0+a_3$

In other words, each of the parity-check bits handles 3 out of the 4 bits of the dataword. The total number of 1s in each 4-bit combination (3 dataword bits and 1 parity bit) must be even. We are not saying that these three equations are unique; any three equations that involve 3 of the 4 bits in the dataword and create independent equations (a combination of two cannot create the third) are valid. The checker in the decoder creates a 3-bit syndrome (s2s1s0) in which each bit is the parity check for 4 out of the 7 bits in the received codeword:

So == *b2*+ b1 + bo+ qo
Sl =*b3* + *b2*+ b1 + ql
S2 == b1 +*bo*+*b3* +*q2*

The equations used by the checker are the same as those used by the generator with the parity-check bits added to the right-hand side of the equation. The 3-bit syndrome creates eight different bit patterns (000 to 111) that can represent eight different conditions.

These conditions define a lack of error or an error in 1 of the 7 bits of the received codeword, as shown in Table 10.5

Table 10.5 *Logical decision made by the correction logic analyzer a/the decoder*

Syndrome	000	001	010	011	100	101	110	111
Error	None	q_0	q1	b_2	*q2*	b_O	b_3	b_1

Note that the generator is not concerned with the four cases shaded in Table 10.5 because there is either no error or an error in the parity bit. In the other four cases, 1 of the bits must be flipped (changed from 0 to 1 or 1 to 0) to find the correct dataword. The syndrome values in Table 10.5 are based on the syndrome bit calculations. For example, if *qo* is in error, *So* is the only bit affected; the syndrome, therefore, is 001. If *b2* is in error, *So* and *s1* are the bits affected; the syndrome, therefore is O1l. Similarly, if bI is in error, all 3 syndrome bits are affected and the syndrome is 111.

There are two points we need to emphasize here. First, if two errors occur during transmission, the created dataword might not be the right one. Second, if we want to use the above code for error detection, we need a different design.

Example

Let us trace the path of three datawords from the sender to the destination:

- The dataword 0100 becomes the codeword 0100011. The codeword 01 00011 is received. The syndrome is 000 (no error), the final dataword is 0100.
- The dataword 0111 becomes the codeword 0111001. The codeword 0011001 is received. The syndrome is 011. According to Table 10.5, *b2* is in error. After flipping *b2* (changing the
- 1 to 0), the final dataword is 0111.
- The dataword 1101 becomes the codeword 1101000. The codeword 0001000 is received (two errors). The syndrome is 101, which means that *bo*is in error. After flipping *ba*, we get 0000, the wrong dataword. This shows that our code cannot correct two errors.