Module-3

BANDWIDTH UTILIZATION

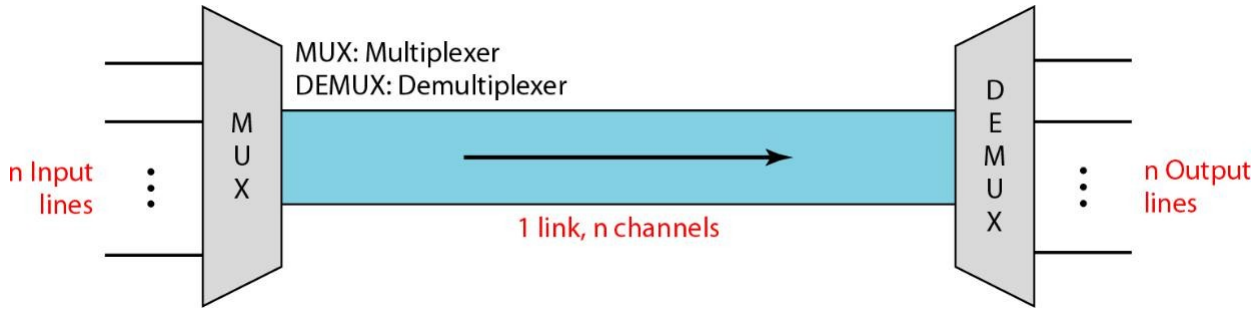
Bandwidth utilization is the wise use of available bandwidth to achieve specific goals. Efficiency can be achieved by multiplexing; privacy and anti-jamming can be achieved by spreading.

Multiplexing

Multiplexing is the set of techniques that allows the simultaneous transmission of multiple signals across a single data link.

In a multiplexed system, n lines share the bandwidth of one link.

Below figure shows the basic format of a multiplexed system.

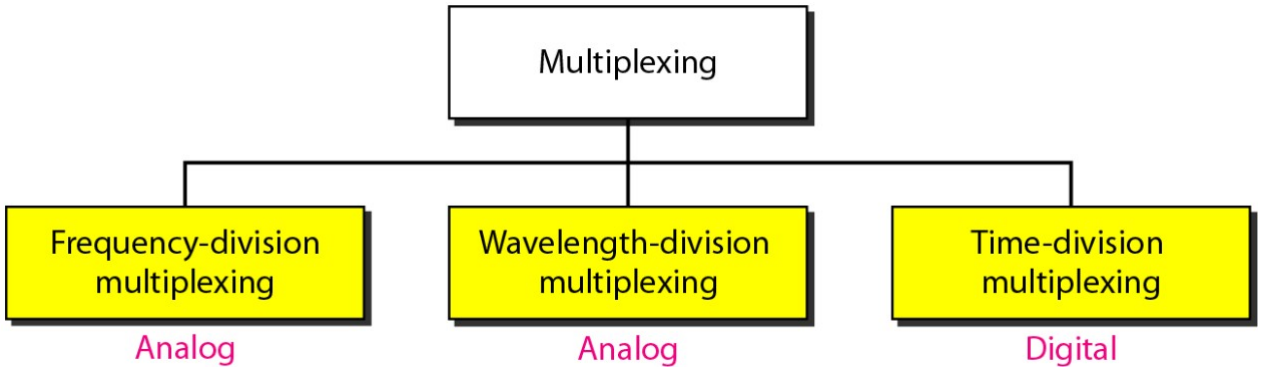


The lines on the left direct their transmission streams to a multiplexer (MUX), which combines them into a single stream (many-to one).

At the receiving end, that stream is fed into a demultiplexer (DEMUX), which separates the stream back into its component transmissions (one-to-many) and directs them to their corresponding lines.

In the figure, the word link refers to the physical path. The word channel refers to the portion of a link that carries a transmission between a given pair of lines. One link can have many (n) channels.

There are three basic multiplexing techniques: frequency-division multiplexing, wavelength-division multiplexing, and time-division multiplexing.



Frequency-Division Multiplexing

Frequency-division multiplexing (FDM) is an analog technique that can be applied when the bandwidth of a link (in hertz) is greater than the combined bandwidths of the signals to be transmitted.

In FDM, signals generated by each sending device modulate different carrier frequencies.

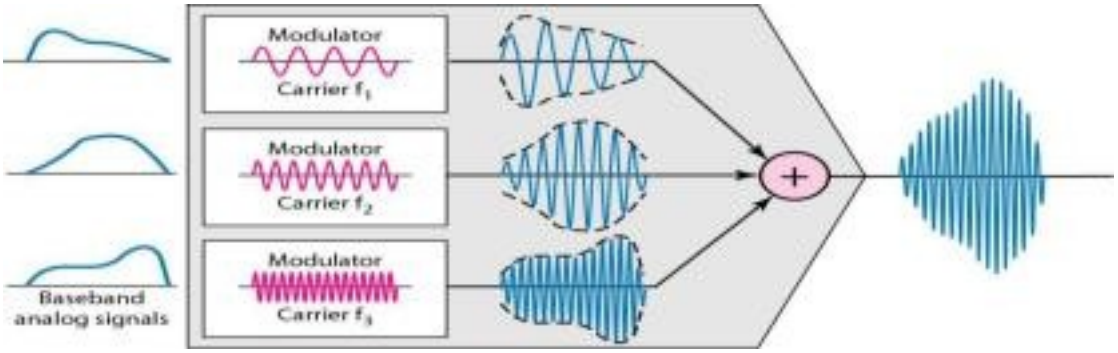
These modulated signals are then combined into a single composite signal that can be transported by the link.

Carrier frequencies are separated by sufficient bandwidth to accommodate the modulated signal. These bandwidth ranges are the channels through which the various signals travel. Channels can be separated by strips of unused bandwidth-guard bands-to prevent signals from overlapping.

In addition, carrier frequencies must not interfere with the original data frequencies.

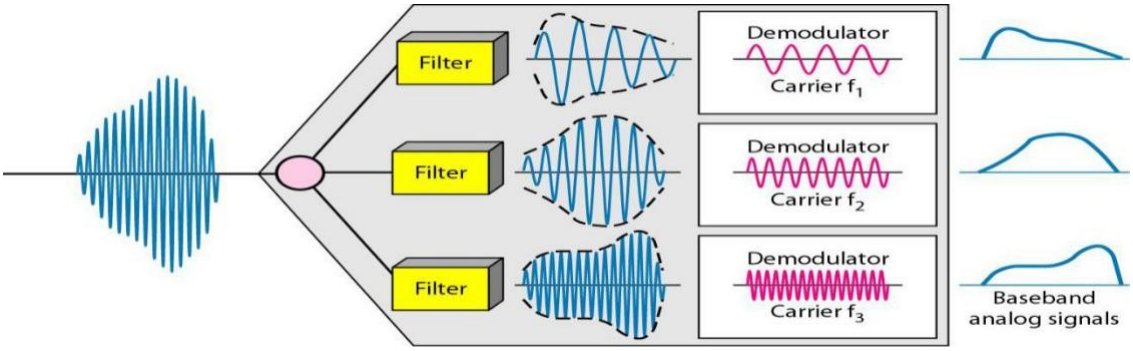
Multiplexing Process

Each source generates a signal of a similar frequency range. Inside the multiplexer, these similar signals modulates different carrier frequencies *(f1,f2 and f3)*. The resulting modulated signals are then combined into a single composite signal that is sent out over a media link that has enough bandwidth to accommodate it.



Demultiplexing Process

The demultiplexer uses a series of filters to decompose the multiplexed signal into its constituent component signals. The individual signals are then passed to a demodulator that separates them from their carriers and passes them to the output lines.

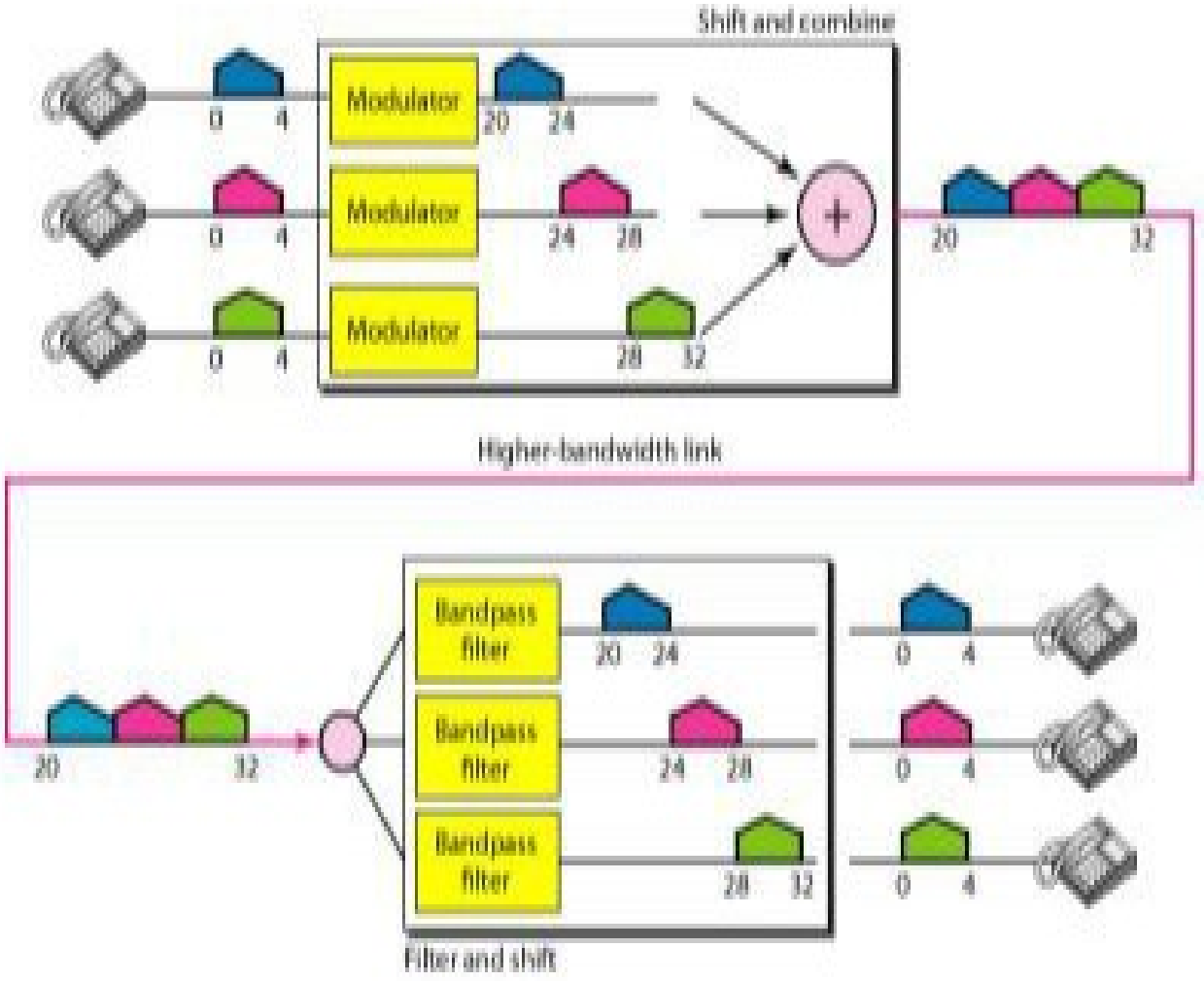


Examples:

1. Assume that a voice channel occupies a bandwidth of 4 kHz. We need to combine three voice channels into a link with a bandwidth of 12 kHz, from 20 to 32 kHz. Show the configuration, using the frequency domain. Assume there are no guard bands.

Solution

We shift (modulate) each of the three voice channels to a different bandwidth, as shown in Figure

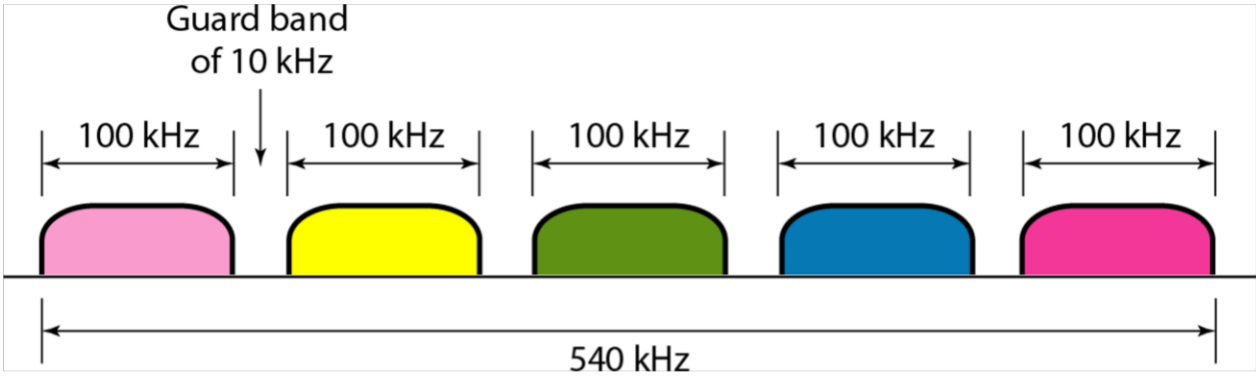


We use the 20- to 24-kHz bandwidth for the first channel, the 24- to 28-kHz bandwidth for the second channel, and the 28- to 32-kHz bandwidth for the third one. Then we combine them. At the receiver, each channel receives the entire signal, using a filter to separate out its own signal. The first channel uses a filter that passes frequencies between 20 and 24 kHz and filters out (discards) any other frequencies. The second channel uses a filter that passes frequencies between 24 and 28 kHz, and the third channel uses a filter that passes frequencies between 28 and 32 kHz. Each channel then shifts the frequency to start from zero.

2. Five channels, each with a 100-kHz bandwidth, are to be multiplexed together. What is the minimum bandwidth of the link if there is a need for a guard band of 10kHz between the channels to prevent interference?

Solution:

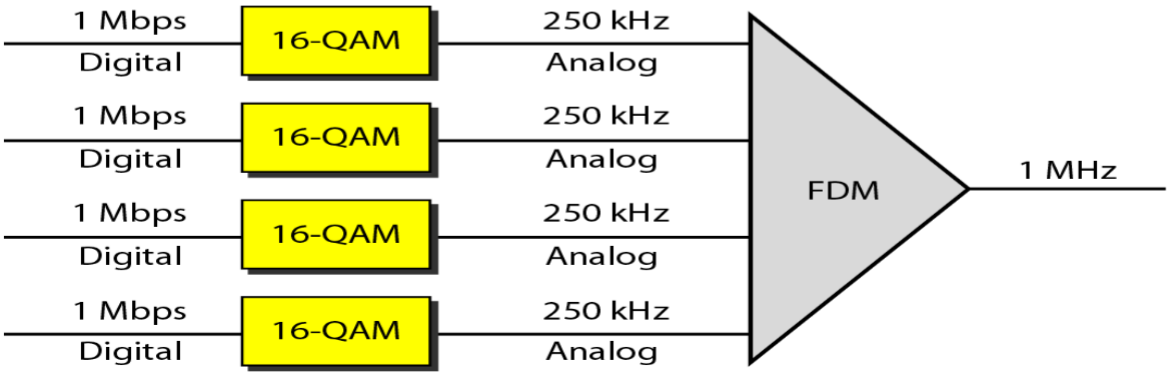
For five channels, we need at least four guard bands. This means that the required bandwidth is at least 5 x 100 + 4 x 10 =540 kHz, as shown in Figure



3) Four data channels (digital), each transmitting at I Mbps, use a satellite channel of I MHz. Design an appropriate configuration, using FDM.

Solution

The satellite channel is analog. We divide it into four channels, each channel having a 250-kHz bandwidth. Each digital channel of 1 Mbps is modulated such that each 4 bits is modulated to 1 Hz. One solution is 16-QAM modulation. Figure shows one possible configuration.



Application:

The Analog Carrier System

To maximize the efficiency of infrastructure, telephone companies have traditionally multiplexed signals from lower-bandwidth lines onto higher-bandwidth lines. In this way, many switched or leased lines can be combined into fewer but bigger channels. For analog lines, FDM is used.

One of these hierarchical systems used by AT&T is made up of groups, supergroups, master groups, and jumbo groups.

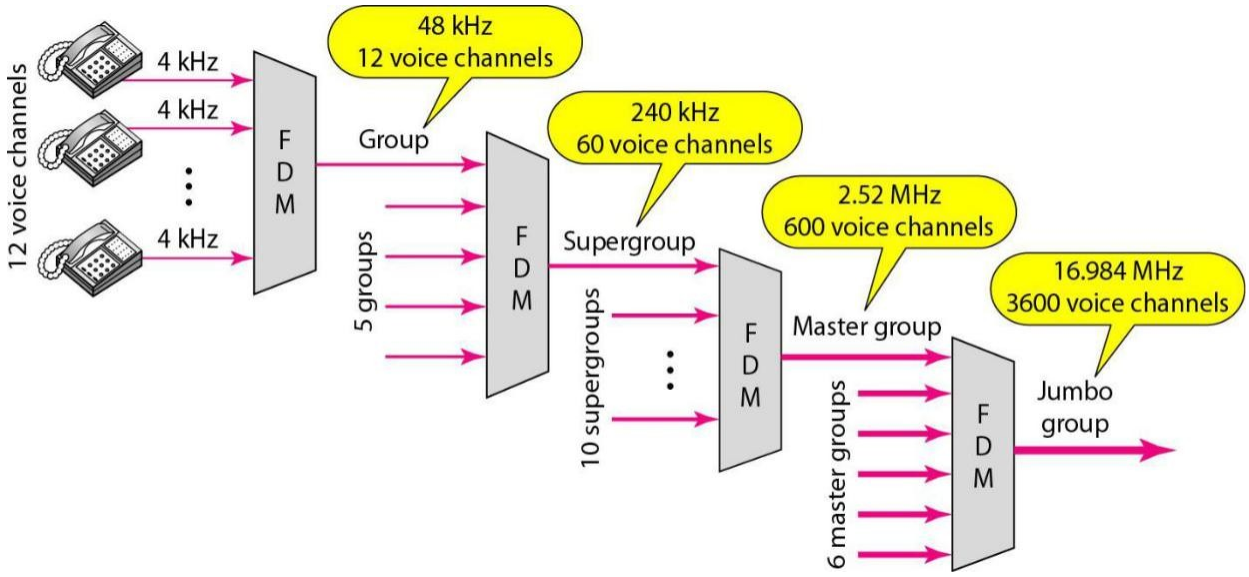
In this analog hierarchy, 12 voice channels are multiplexed onto a higher-bandwidth line to create a group. A group has 48 kHz of bandwidth and supports 12 voice channels.

At the next level, up to five groups can be multiplexed to create a composite signal called a supergroup. A supergroup has a bandwidth of 240 kHz and supports up to 60 voice channels. Supergroups can be made up of either five groups or 60 independent voice channels.

At the next level, 10 supergroups are multiplexed to create a master group. A master group must have 2.40 MHz of bandwidth, but the need for guard bands between the supergroups increases the necessary bandwidth to 2.52 MHz. Master groups support up to 600 voice channels.

Finally, six master groups can be combined into a jumbo group. A jumbo group must have

15.12 MHz (6 x 2.52 MHz) but is augmented to 16.984 MHz to allow for guard bands between the master groups.



➢ ***Other Applications of FDM***

AM and FM radio broadcasting.

Radio uses the air as the transmission medium. A special band from 530 to 1700 kHz is assigned to AM radio. All radio stations need to share this band. Each AM station needs 10 kHz of bandwidth. Each station uses a different carrier frequency, which means it is shifting its signal and multiplexing. The signal that goes to the air is a combination of signals. A receiver receives all these signals, but filters (by tuning) only the one which is desired. Without multiplexing, only one AM station could broadcast to the common link, the air.

However, we need to know that there is physical multiplexer or demultiplexer here. Multiplexing is done at the data link layer.

The situation is similar in FM broadcasting. However, FM has a wider band of 88 to 108 MHz because each station needs a bandwidth of 200 kHz.

➢ ***Television broadcasting.***

Each TV channel has its own bandwidth of 6 MHz.

➢ ***The first generation of cellular telephones.***

Each user is assigned two 30-kHz channels, one for sending voice and the other for receiving. The voice signal, which has a bandwidth of 3 kHz (from 300 to 3300 Hz), is modulated by using FM.

Example

The Advanced Mobile Phone System (AMPS) uses two bands. The first band of 824 to 849 MHz is used for sending, and 869 to 894 MHz is used for receiving. Each user has a bandwidth of 30 kHz in each direction. The 3-kHz voice is modulated using FM, creating 30 kHz of modulated signal. How many people can use their cellular phones simultaneously?

Solution:

Each band is 25 MHz. If we divide 25 MHz by 30 kHz, we get 833.33. In reality, the band is divided into 832 channels. Of these, 42 channels are used for control, which means only 790 channels are available for cellular phone users.

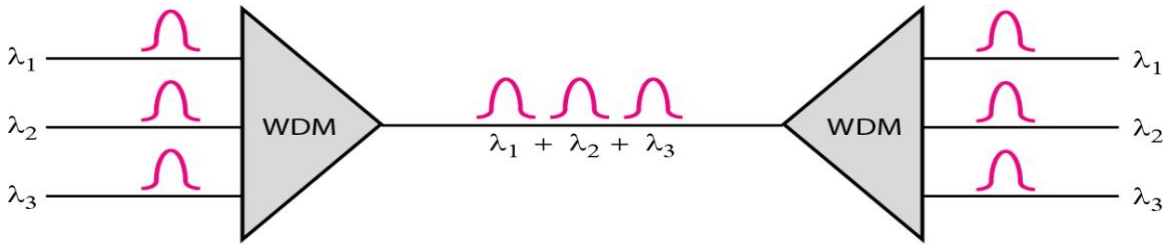
Wavelength-Division Multiplexing

Wavelength-division multiplexing (WDM) is designed to use the high-data-rate capability of fiber-optic cable.

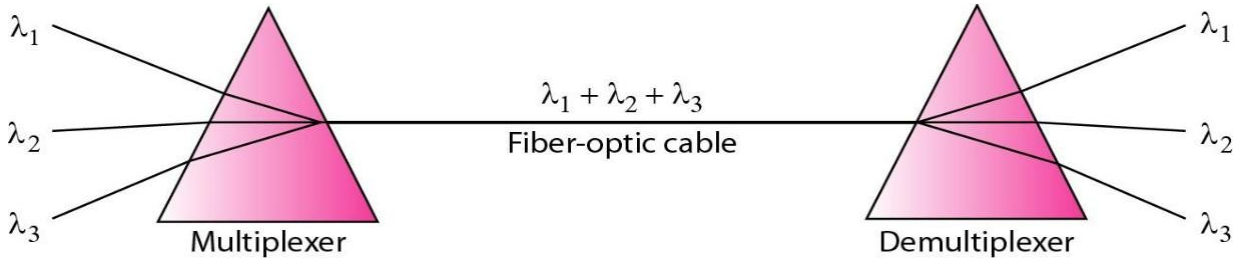
The optical fiber data rate is higher than the data rate of metallic transmission cable. Using a fiber-optic cable for one single line wastes the available bandwidth. Multiplexing allows us to combine several lines into one.

WDM is conceptually the same as FDM, except that the multiplexing and demultiplexing involve optical signals transmitted through fiber-optic channels

Very narrow bands of light from different sources are combined to make a wider band of light. At the receiver, the signals are separated by the demultiplexer.



The combining and splitting of light sources are easily handled by a prism. Prism bends a beam of light based on the angle of incidence and the frequency. Using this technique, a multiplexer can be made to combine several input beams of light, each containing a narrow band of frequencies, into one output beam of a wider band of frequencies. A demultiplexer can also be made to reverse the process.



One application of WDM is the SONET network in which multiple optical fiber lines are multiplexed and demultiplexed.

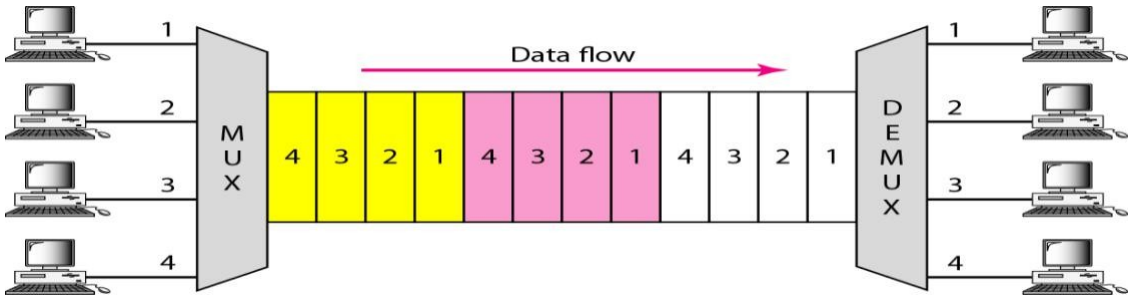
A new method, called dense WDM (DWDM), can multiplex a very large number of channels by spacing channels very close to one another. It achieves even greater efficiency.

Synchronous Time-Division Multiplexing

Time-division multiplexing (TDM) is a digital process that allows several connections to share the high bandwidth of a link.

Instead of sharing a portion of the bandwidth as in FDM, time is shared. Each connection occupies a portion of time in the link.

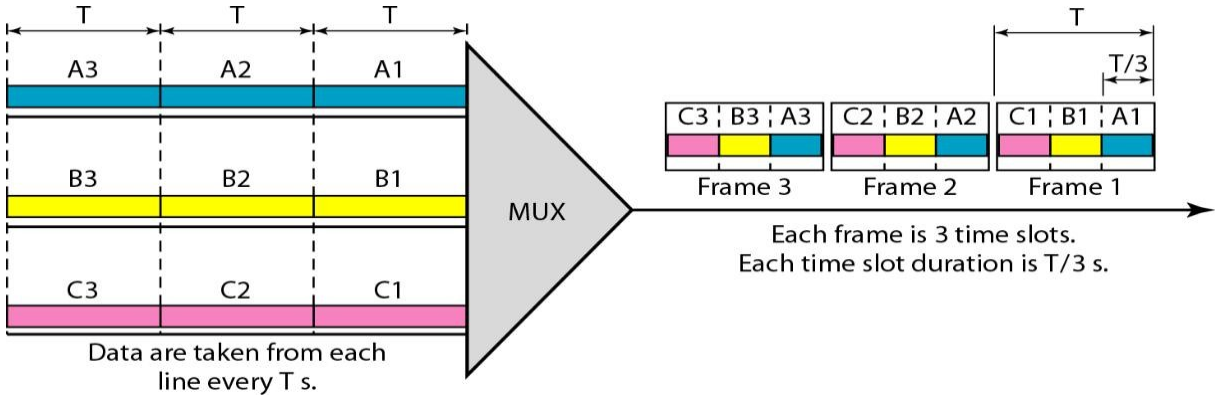
TDM is a digital multiplexing technique for combining several low-rate channels into one high-rate one.



TDM is divided into two different schemes: synchronous and statistical

➢ ***Time Slots and Frames***

In synchronous TDM, the data flow of each input connection is divided into units, where each input occupies one input time slot. A unit can be 1 bit, one character, or one block of data. Each input unit becomes one output unit and occupies one output time slot. However, the duration of an output time slot is *n* times shorter than the duration of an input time slot. If an input time slot is *T* s, the output time slot is *Tn* s, where *n* is the number of connections.



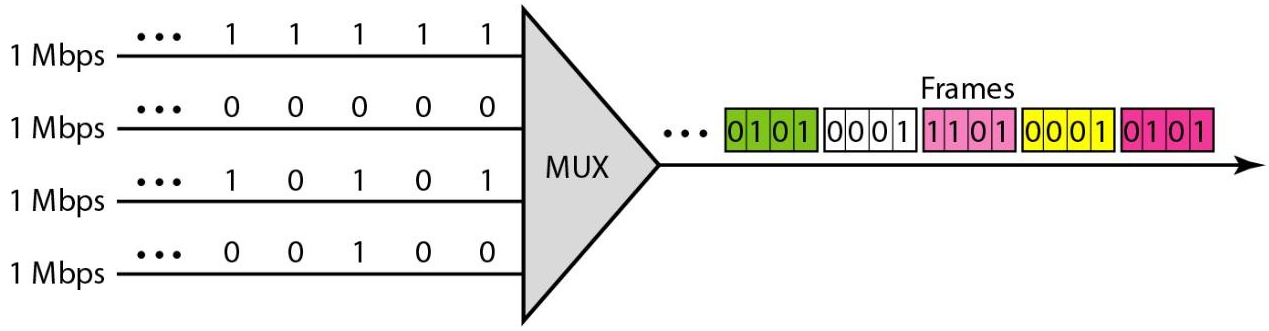
In synchronous TDM, a round of data units from each input connection is collected into a frame. If we have n connections, a frame is divided into n time slots and one slot is allocated for each unit, one for each input line. If the duration of the input unit is T, the duration of each slot is Tin and the duration of each frame is T.

In synchronous TDM, the data rate of the link is n times faster, and the unit duration is n times shorter.

Time slots are grouped into frames. A frame consists of one complete cycle of time slots, with one slot dedicated to each sending device. In a system with *n* input lines, each frame has *n* slots, with each slot allocated to carrying data from a specific input line.

Example:

Figure shows synchronous TOM with a data stream for each input and one data stream for the output. The unit of data is 1 bit. Find (a) the input bit duration, (b) the output bit duration, (c) the output bit rate, and (d) the output frame rate.



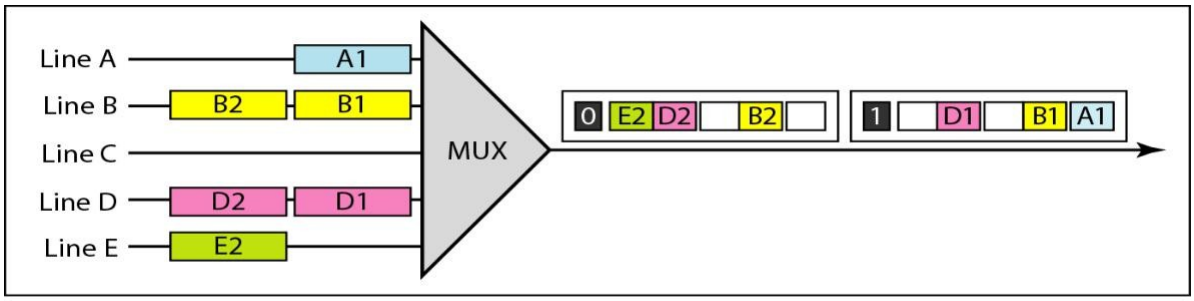
Solution

- The input bit duration is the inverse of the bit rate: 1/1 Mbps = 1 μs.
- The output bit duration is one-fourth of the input bit duration, or 1/4μs.
- The output bit rate is the inverse of the output bit duration or 1/4 μs, or 4 Mbps. This can also be deduced from the fact that the output rate is 4 times as fast as any input rate; so the output rate =4 x 1 Mbps =4 Mbps.
- The frame rate is always the same as any input rate. So the frame rate is 1,000,000 frames per second. Because we are sending 4 bits in each frame, we can verify the result of the previous question by multiplying the frame rate by the number of bits per frame.

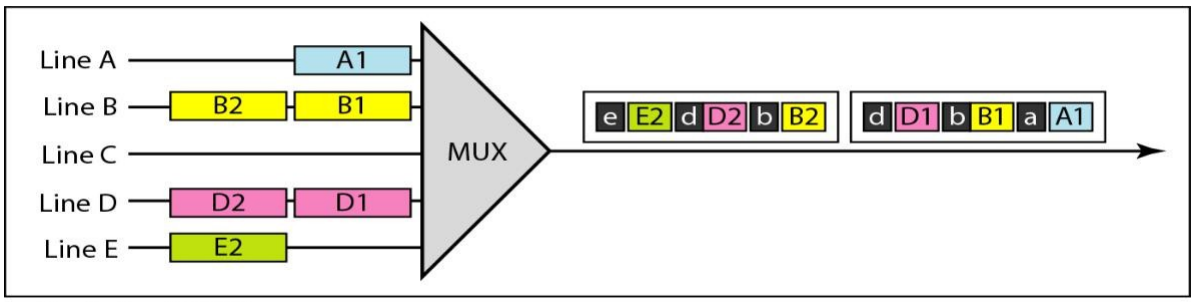
Statistical Time-Division Multiplexing

In statistical time-division multiplexing, slots are dynamically allocated to improve bandwidth efficiency. Only when an input line has a slot's worth of data to send is it given a slot in the output frame. In statistical multiplexing, the number of slots in each frame is less than the number of input lines. The multiplexer checks each input line in round robin fashion; it allocates

a slot for an input line if the line has data to send; otherwise, it skips the line and checks the next line.



a. Synchronous TDM



b. Statistical TDM

Spread Spectrum

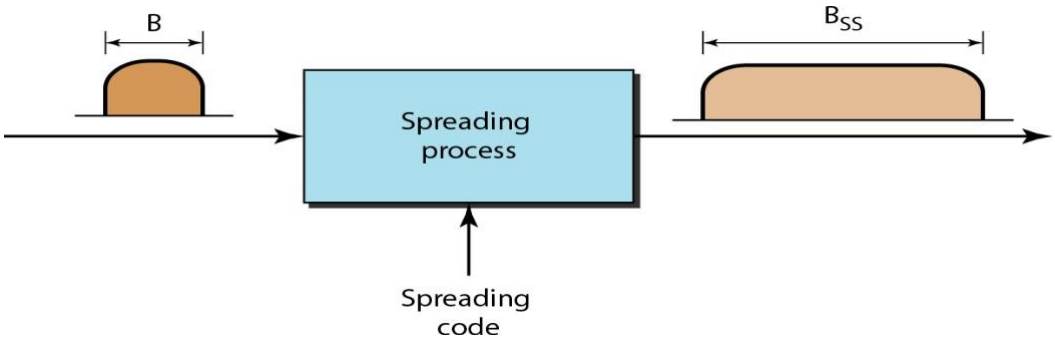
Multiplexing combines signals from several sources to achieve bandwidth efficiency; the available bandwidth of a link is divided between the sources.

In spread spectrum, we also combine signals from different sources to fit into a larger bandwidth.

Spread spectrum is designed to be used in wireless applications (LANs and WANs).

In wireless applications, all stations use air (or a vacuum) as the medium for communication. Stations must be able to share this medium without interception by an eavesdropper and without being subject to jamming from a malicious intruder (in military operations, for example).

If the required bandwidth for each station is B, spread spectrum expands it to Bss such that Bss » B. The expanded bandwidth allows the source to wrap its message in a protective envelope for a more secure transmission.



Spread spectrum achieves its goals through two principles:

1. The bandwidth allocated to each station needs to be, by far, larger than what is needed. This allows redundancy.
2. The expanding of the original bandwidth *B* to the bandwidth Bss must be done by a process that is independent of the original signal. In other words, the spreading process occurs after the signal is created by the source.

After the signal is created by the source, the spreading process uses a spreading code and spreads the bandwidth. The figure shows the original bandwidth B and the spreaded bandwidth Bss. The spreading code is a series of numbers that look random, but are actually a pattern.

There are two techniques to spread the bandwidth:

- Frequency hopping spread spectrum (FHSS)
- Direct sequence spread spectrum (DSSS)

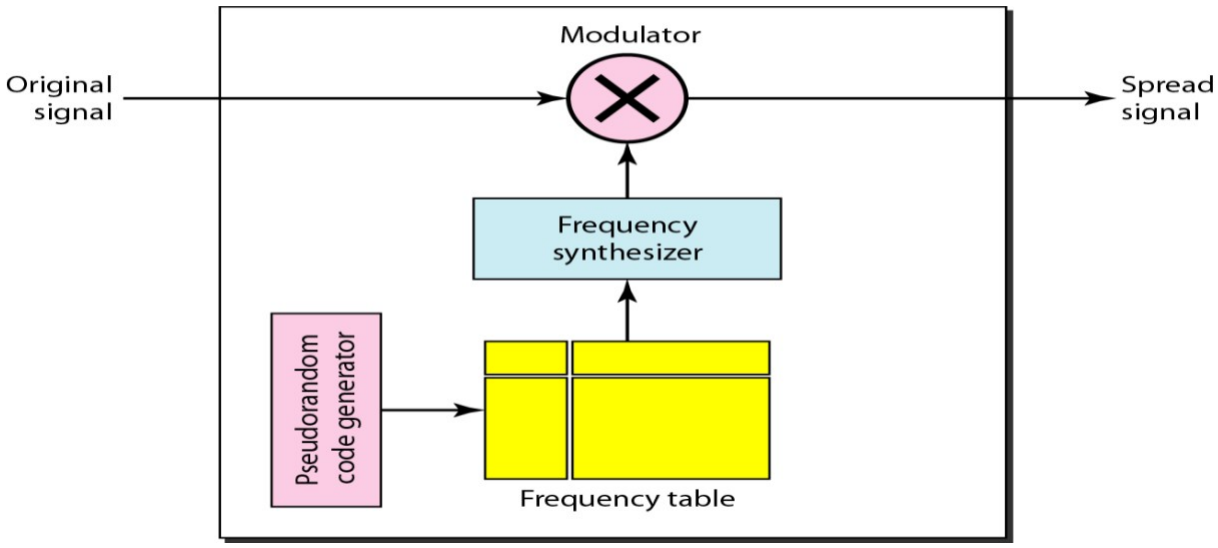
Frequency Hopping Spread Spectrum (FHSS)

The frequency hopping spread spectrum (FHSS) technique uses *M* different carrier frequencies that are modulated by the source signal.

At one moment, the signal modulates one carrier frequency; at the next moment, the signal modulates another carrier frequency. Although the modulation is done using one carrier frequency at a time, *M* frequencies are used in the long run.

The bandwidth occupied by a source after spreading is *BpHSS »B*.

The general layout for FHSS is shown below:

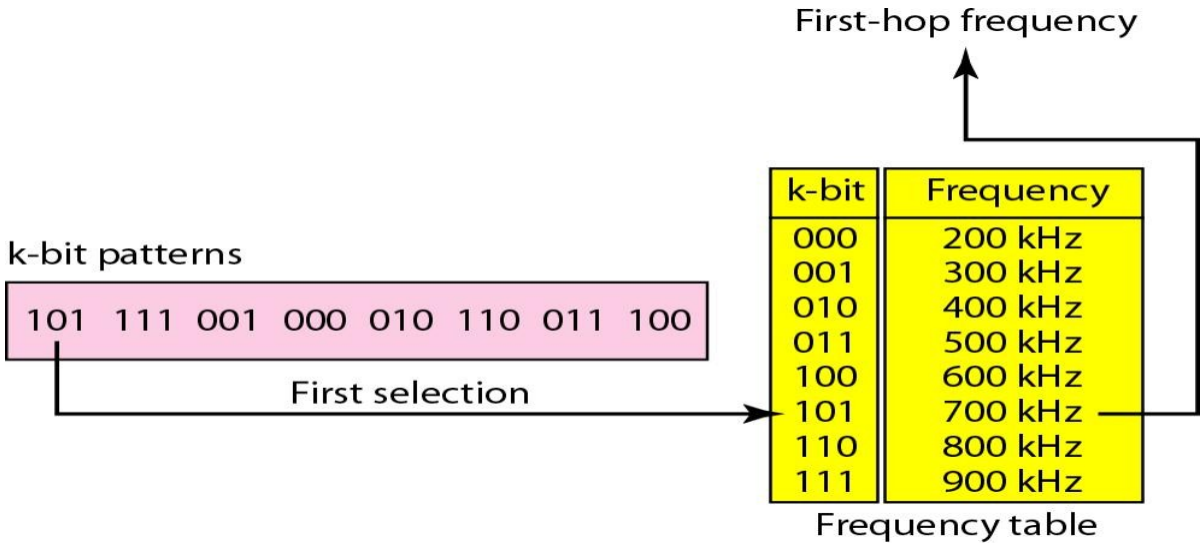


A pseudorandom code generator, called pseudorandom noise (PN), creates a k-bit pattern for every hopping period *Th*.

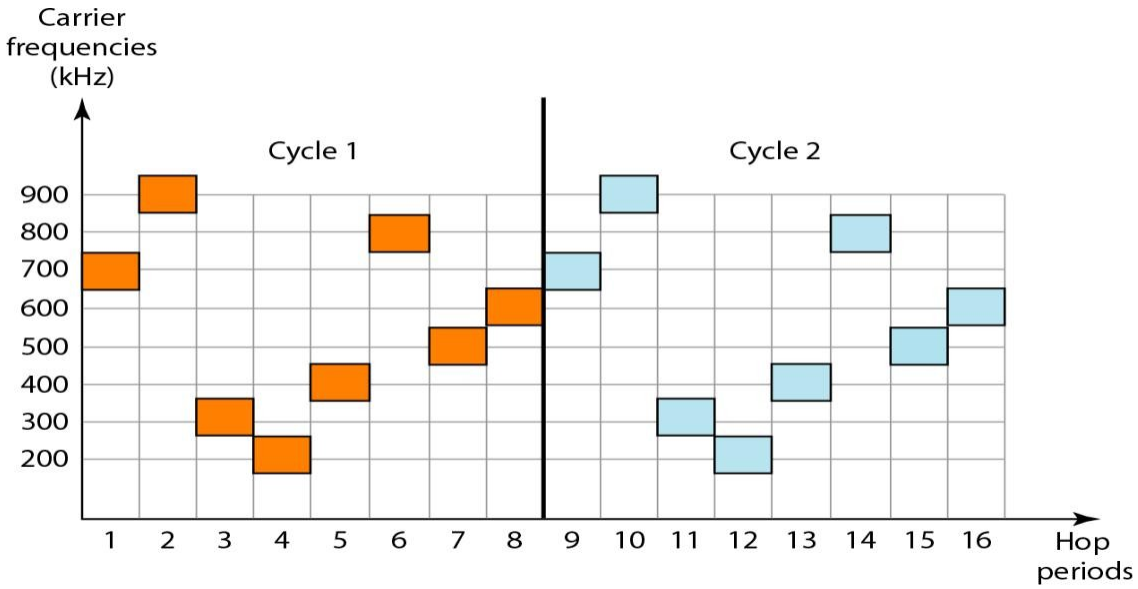
The frequency table uses the pattern to find the frequency to be used for this hopping period and passes it to the frequency synthesizer.

The frequency synthesizer creates a carrier signal of that frequency, and the source signal modulates the carrier signal.

Suppose we have decided to have eight hopping frequencies. This is extremely low for real applications and is just for illustration. In this case, M is 8 and k is 3. The pseudorandom code generator will create eight different 3-bit patterns. These are mapped to eight different frequencies in the frequency table.

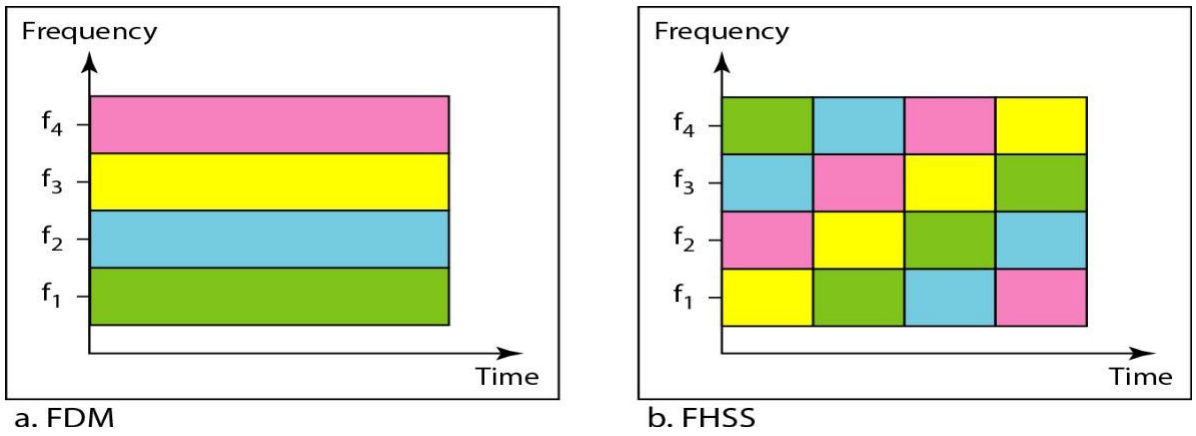


The pattern for this station is 101, 111, 001, 000, 010, all, 100. Note that the pattern is pseudorandom it is repeated after eight hoppings. This means that at hopping period 1, the pattern is 101. The frequency selected is 700 kHz; the source signal modulates this carrier frequency. The second k-bit pattern selected is 111, which selects the 900-kHz carrier; the eighth pattern is 100, the frequency is 600 kHz. After eight hoppings, the pattern repeats, starting from 101 again. Figure shows how the signal hops around from carrier to carrier. We assume the required bandwidth of the original signal is 100 kHz.



Bandwidth Sharing

If the number of hopping frequencies is *M,* we can multiplex *M* channels into one by using the same Bss bandwidth. This is possible because a station uses just one frequency in each hopping period; *M* - 1 other frequencies can be used by other *M* - 1 stations. In other words, *M* different stations can use the same Bss if an appropriate modulation technique such as multiple FSK (MFSK) is used.

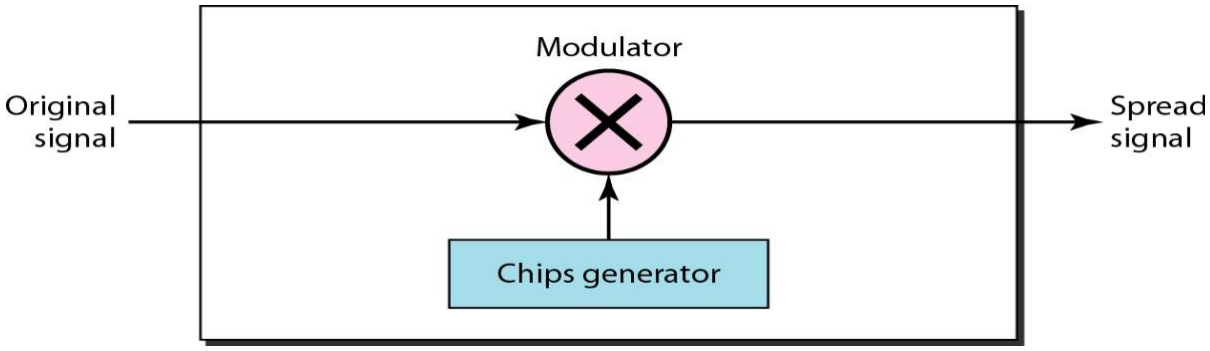


Above Figure shows an example of four channels using FDM and four channels using FHSS. In FDM, each station uses 11M of the bandwidth, but the allocation is fixed; in FHSS, each station uses 11M of the bandwidth, but the allocation changes hop to hop.

Direct Sequence Spread Spectrum

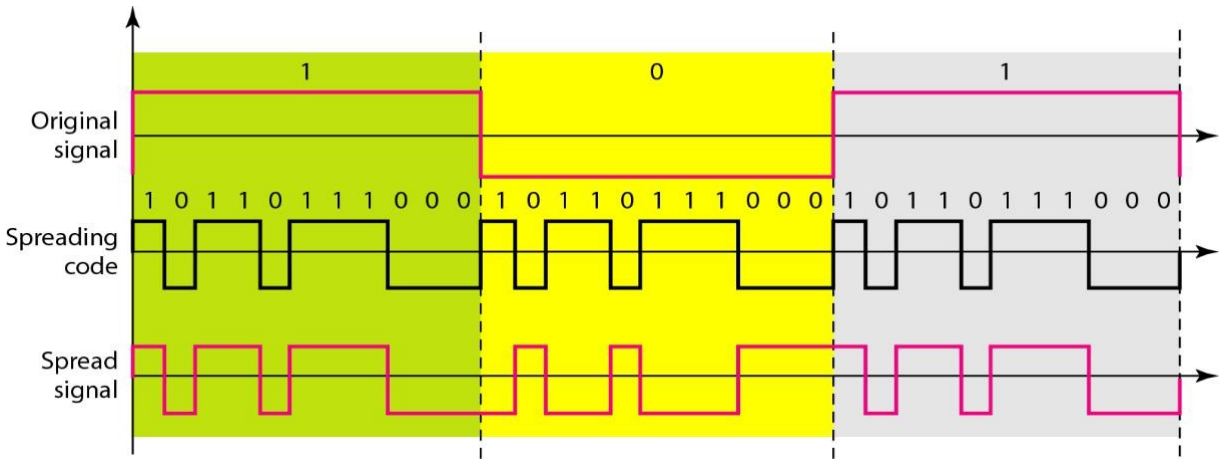
The direct sequence spread spectrum (DSSS) technique also expands the bandwidth of the original signal, but the process is different.

In DSSS, we replace each data bit with 11 bits using a spreading code. In other words, each bit is assigned a code of 11 bits, called **chips**, where the chip rate is 11 times that of the data bit.



As an example, let us consider the sequence used in a wireless LAN, the famous Barker sequence where 11 is 11. We assume that the original signal and the chips in the chip generator use polar NRZ encoding. Below Figure shows the chips and the result of multiplying the original data by the chips to get the spread signal. In Figure, the spreading code is 11 chips having the pattern 10110111000 (in this case). If the original signal rate is N, the rate of the spread signal is

l1N. This means that the required bandwidth for the spread signal is 11 times larger than the bandwidth of the original signal. The spread signal can provide privacy if the intruder does not know the code. It can also provide immunity against interference if each station uses a different code.

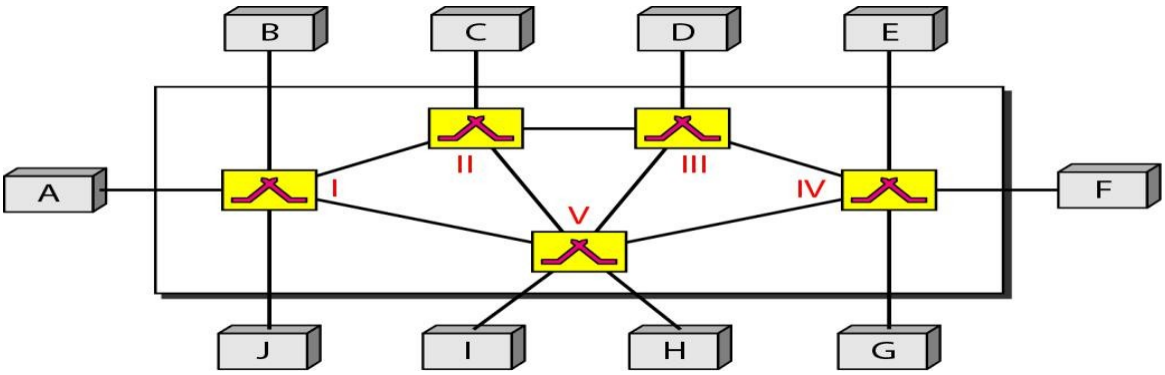


Switching

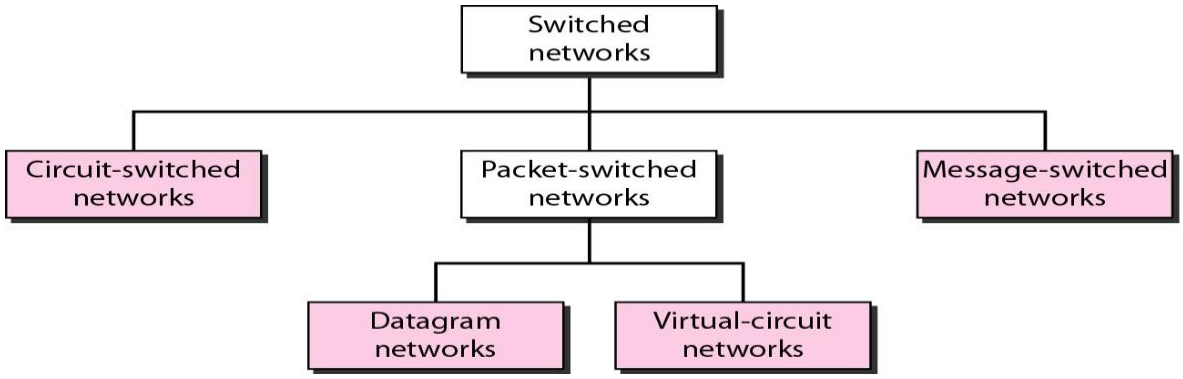
A switched network consists of a series of interlinked nodes, called switches.

Switches are devices capable of creating temporary connections between two or more devices linked to the switch.

In a switched network, some of these nodes are connected to the end systems. Others are used only for routing.



Switched networks can be divided into three broad categories: circuit-switched networks, packet-switched networks, and message-switched. Packet-switched networks can further be divided into two subcategories-virtual-circuit networks and datagram networks.



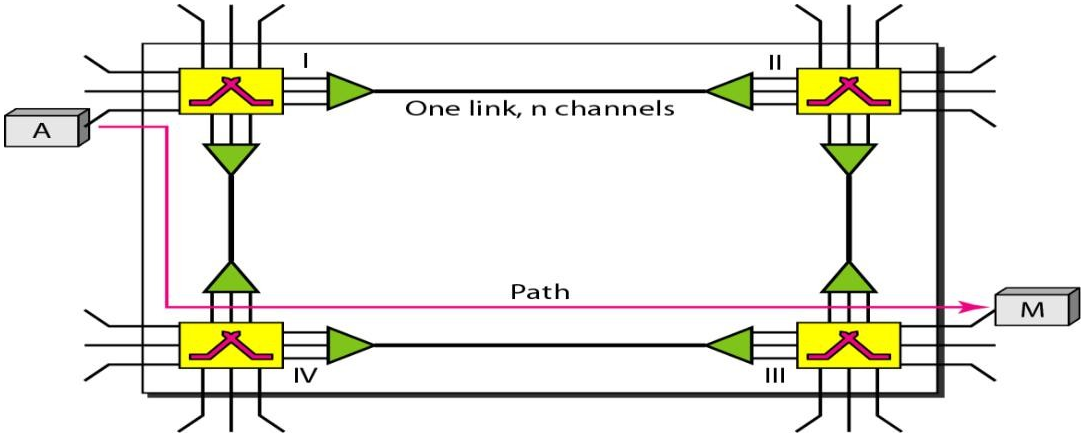
• Circuit Switched Networks

A circuit-switched network consists of a set of switches connected by physical links.

A connection between two stations is a dedicated path made of one or more links. However, each connection uses only one dedicated channel on each link.

Each link is normally divided into *n* channels by using FDM or TDM.

Below Figure shows a trivial circuit-switched network with four switches and four links. Each link is divided into n (n is 3 in the figure) channels by using FDM or TDM.



Circuit switching takes place at the physical layer.

Before starting communication, the stations must make a reservation for the resources to be used during the communication. These resources, such as channels (bandwidth in FDM and time slots in TDM), switch buffers, switch processing time, and switch input/output ports, must remain dedicated during the entire duration of data transfer until the teardown phase.

1) Data transferred between the two stations are not packetized. The data are a continuous flow sent by the source station and received by the destination station, although there may be periods of silence.

2) There is no addressing involved during data transfer. The switches route the data based on their occupied band (FDM) or time slot (TDM).

Three Phases

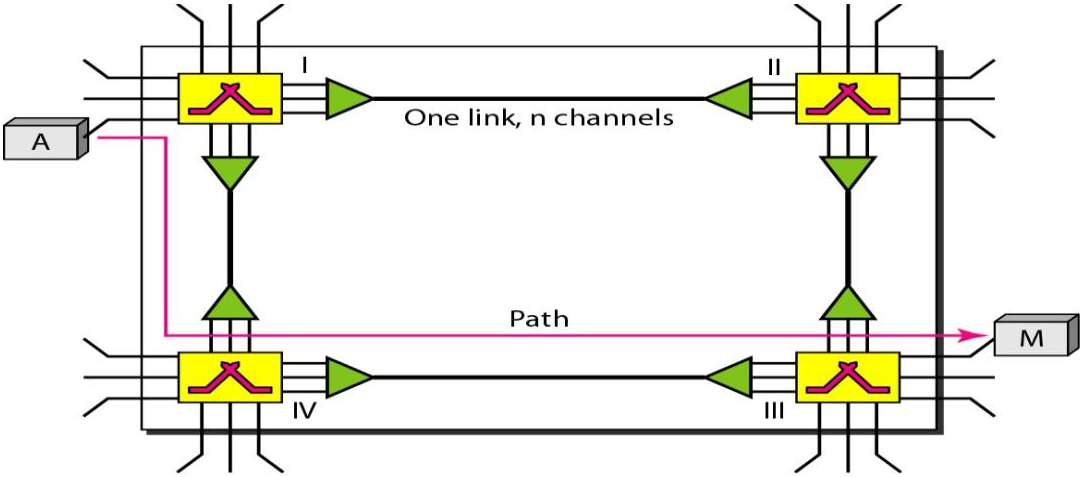
The actual communication in a circuit-switched network requires three phases: connection setup, data transfer, and connection teardown.

1. Setup Phase

Before the two parties can communicate, a dedicated circuit needs to be established. The end systems are normally connected through dedicated lines to the switches, so connection setup means creating dedicated channels between the switches.

In the next step to making a connection, an acknowledgment from system M needs to be sent in the opposite direction to system A. Only after system A receives this acknowledgment is the connection established.

2. Data Transfer Phase



After the establishment of the dedicated circuit (channels), the two parties can transfer data.

3. Teardown Phase

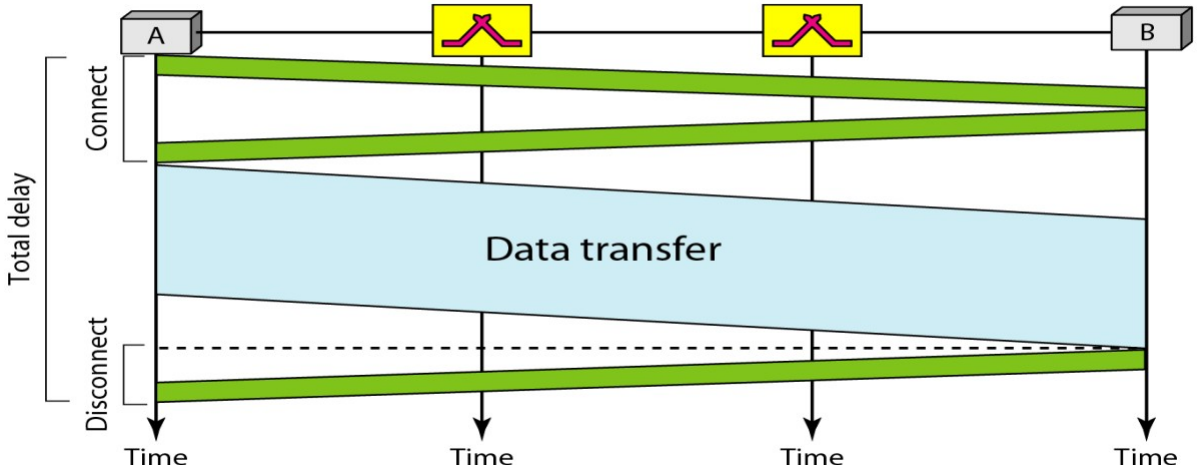
When one of the parties needs to disconnect, a signal is sent to each switch to release the resources.

Efficiency

Circuit-switched networks are not as efficient as the other two types of networks because resources are allocated during the entire duration of the connection. These resources are unavailable to other connections.

Delay

Although a circuit-switched network normally has low efficiency, the delay in this type of network is minimal. During data transfer the data are not delayed at each switch; the resources are allocated for the duration of the connection.



As Figure shows, there is no waiting time at each switch. The total delay is due to the time needed to create the connection, transfer data, and disconnect the circuit.

The delay caused by the setup is the sum of four parts: the propagation time of the source computer request, the request signal transfer time, the propagation time of the acknowledgment from the destination computer, and the signal transfer time of the acknowledgment.

The delay due to data transfer is the sum of two parts: the propagation time and data transfer time, which can be very long.

The third box shows the time needed to tear down the circuit.

Datagram Networks

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.

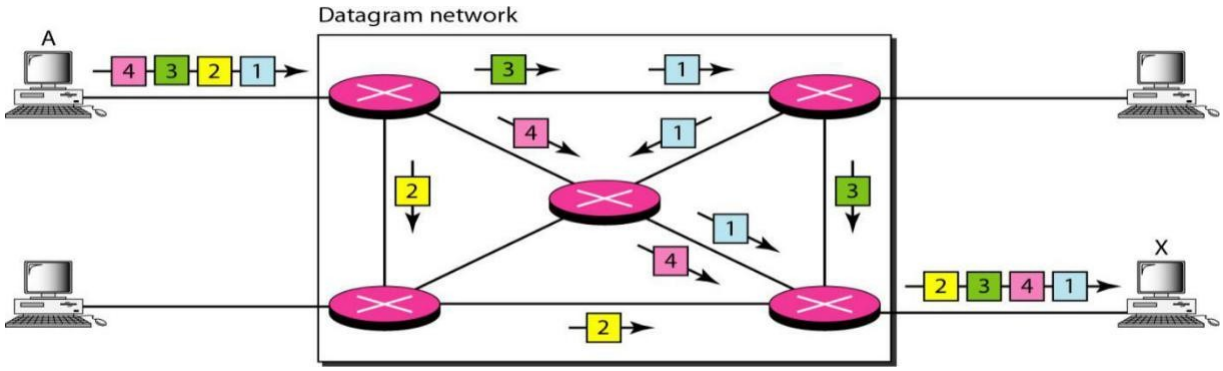
Resources are allocated on demand. The allocation is done on a first come, first-served basis. When a switch receives a packet, no matter what is the source or destination, the packet must wait if there are other packets being processed.

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.

The datagram networks are sometimes referred to as connectionless networks. The term *connectionless* here means that the switch (packet switch) does not keep information about the connection state. There are no setup or teardown phases. Each packet is treated the same by a switch regardless of its source or destination.

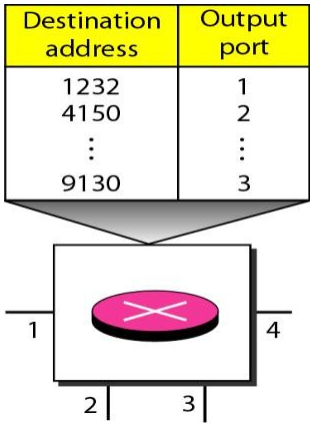
Example: Below Figure shows how the datagram approach is used to deliver four packets from station A to station X. The switches in a datagram network are traditionally referred to as routers.



In this example, all four packets (or datagrams) belong to the same message, but may travel different paths to reach their destination. This is so because the links may be involved in carrying packets from other sources and do not have the necessary bandwidth available to carry all the packets from A to X. This approach can cause the datagrams of a transmission to arrive at their destination out of order with different delays between the packets. Packets may also be lost or dropped because of a lack of resources.

Routing Table

Each switch (or packet switch) has a routing table which is based on the destination address. The routing tables are dynamic and are updated periodically. The destination addresses and the corresponding forwarding output ports are recorded in the tables.



Destination Address

Every packet in a datagram network carries a header that contains the destination address of the packet.

When the switch receives the packet, this destination address is examined; the routing table is consulted to find the corresponding port through which the packet should be forwarded.

The destination address in the header of a packet in a datagram network remains the same during the entire journey of the packet.

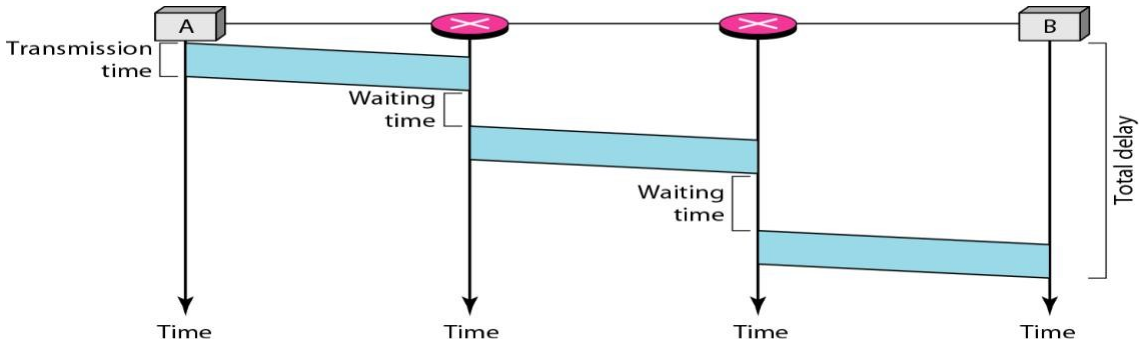
Efficiency

The efficiency of a datagram network is better than that of a circuit-switched network; resources are allocated only when there are packets to be transferred. If a source sends a packet and there is a delay of a few minutes before another packet can be sent, the resources can be reallocated during these minutes for other packets from other sources.

Delay

There may be greater delay in a datagram network than in a virtual-circuit network. Although there are no setup and teardown phases, each packet may experience a wait at a switch before it is forwarded. In addition, since not all packets in a message necessarily travel through the same switches, the delay is not uniform for the packets of a message.

Below figure shows an example of delay in a datagram network for one single packet.



The packet travels through two switches. There are three transmission times (3T), three propagation delays (slopes 3τ of the lines), and two waiting times (W1 + W2)' We ignore the processing time in each switch. The total delay is

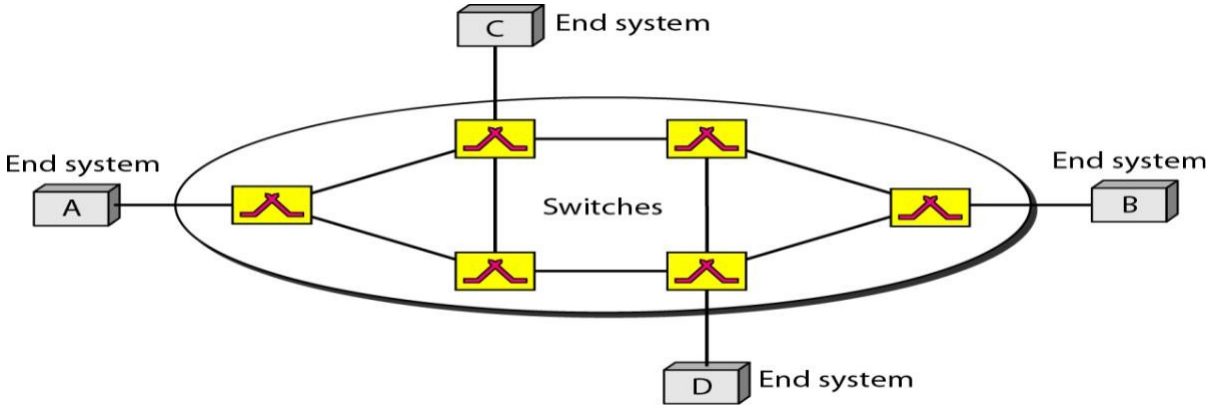
$$\text{Total delay} = 3T + 3\tau + W1 + W2$$

Virtual-Circuit Networks

A virtual-circuit network is a cross between a circuit-switched network and a datagram network. It has some characteristics of both.

- As in a circuit-switched network, there are setup and teardown phases in addition to the data transfer phase.
- Resources can be allocated during the setup phase, as in a circuit-switched network, or on demand, as in a datagram network.
- As in a datagram network, data are packetized and each packet carries an address in the header. However, the address in the header has local jurisdiction, not end-to-end jurisdiction.
- As in a circuit-switched network, all packets follow the same path established during the connection.
- A virtual-circuit network is normally implemented in the data link layer, while a circuit-switched network is implemented in the physical layer and a datagram network in the network layer.

Below Figure is an example of a virtual-circuit network. The network has switches that allow traffic from sources to destinations. A source or destination can be a computer, packet switch, bridge, or any other device that connects other networks.



Addressing

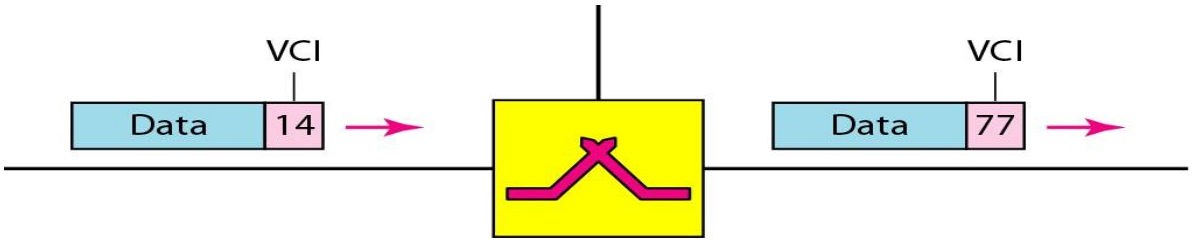
In a virtual-circuit network, two types of addressing are involved: global and local (virtual-circuit identifier).

Global Addressing

A source or a destination needs to have a global address-an address that can be unique in the scope of the network or internationally if the network is part of an international network.

Virtual-Circuit Identifier

The identifier that is actually used for data transfer is called the virtual-circuit identifier (VCI). A VCI is a small number that has only switch scope; it is used by a frame between two switches. When a frame arrives at a switch, it has a VCI; when it leaves, it has a different VCI. Below Figure shows how the VCI in a data frame changes from one switch to another.



Three Phases

Data Transfer Phase

To transfer a frame from a source to its destination, all switches need to have a table entry for this virtual circuit. The table, in its simplest form, has four columns. This means that the switch holds four pieces of information for each virtual circuit that is already set up. Below Figure shows such a switch and its corresponding table.

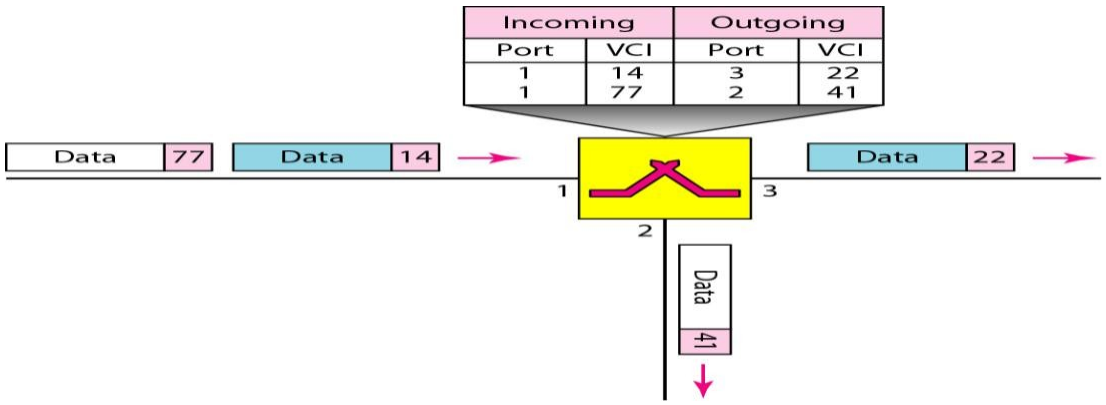
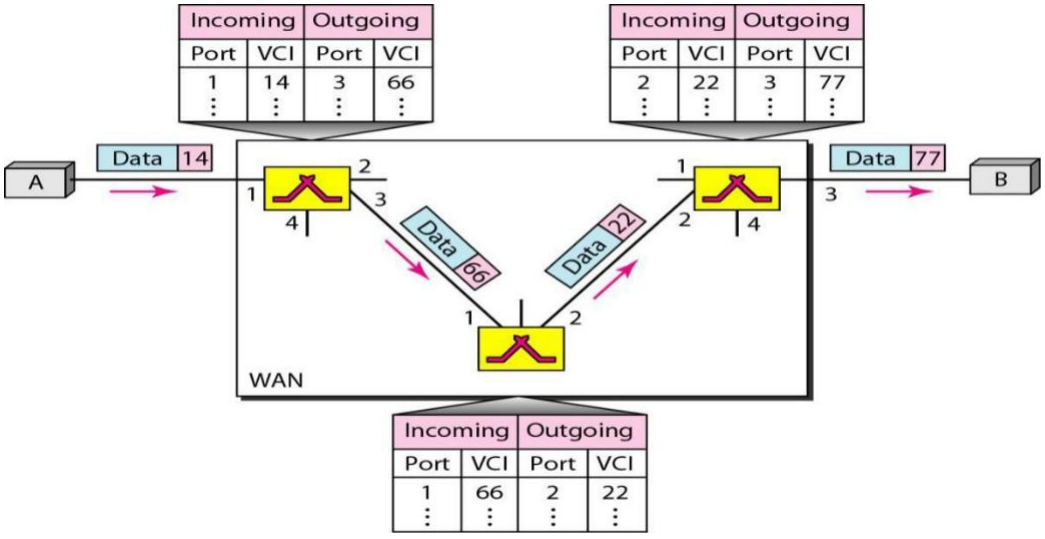


Figure shows a frame arriving at port 1 with a VCI of 14. When the frame arrives, the switch looks in its table to find port 1 and a VCI of 14. When it is found, the switch knows to change the VCI to 22 and send out the frame from port 3.

Below Figure shows how a frame from source A reaches destination B and how its VCI changes during the trip. Each switch changes the VCI and routes the frame.



The data transfer phase is active until the source sends all its frames to the destination.

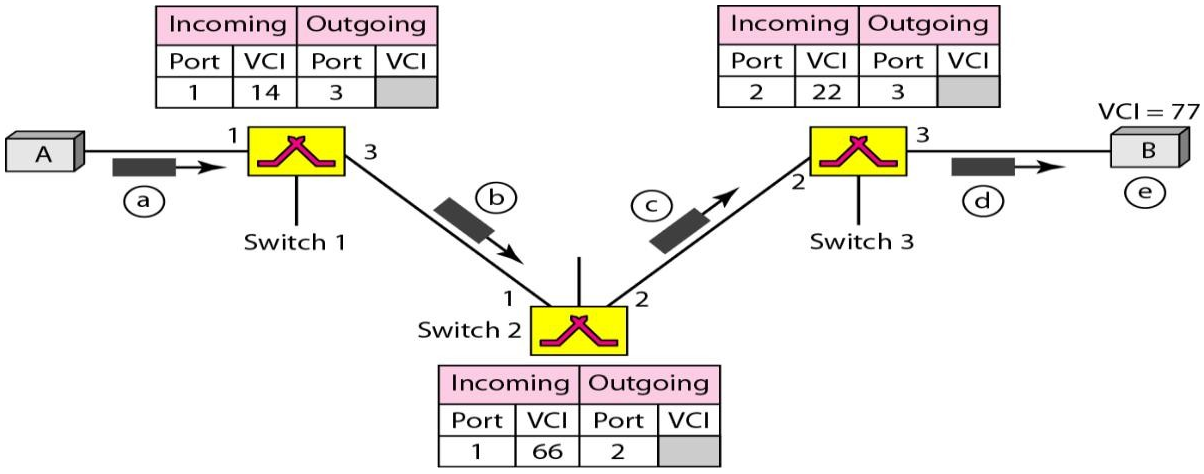
The procedure at the switch is the same for each frame of a message. The process creates a virtual circuit, not a real circuit, between the source and destination.

Setup Phase

In the setup phase, a switch creates an entry for a virtual circuit.

For example, suppose source A needs to create a virtual circuit to B. Two steps are required: the setup request and the acknowledgment.

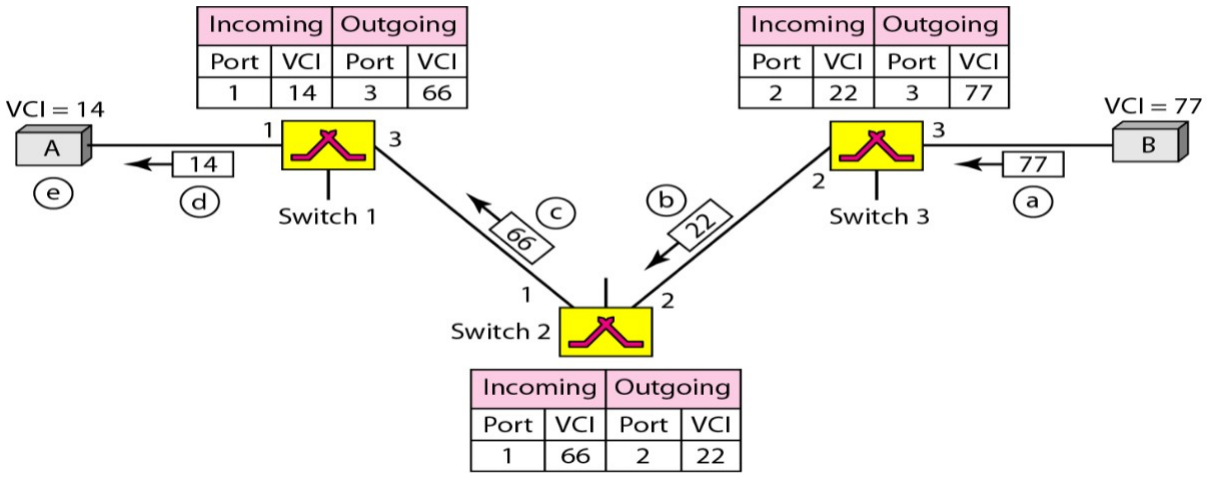
Setup Request: A setup request frame is sent from the source to the destination.



Setup request in a virtual-circuit network

1. Source A sends a setup frame to switch 1.
2. Switch 1 receives the setup request frame. It knows that a frame going from A to B goes out through port 3. The switch, in the setup phase, acts as a packet switch; it has a routing table which is different from the switching table. For the moment, assume that it knows the output port. The switch creates an entry in its table for this virtual circuit, but it is only able to fill three of the four columns. The switch assigns the incoming port (1) and chooses an available incoming VCI (14) and the outgoing port (3). It does not yet know the outgoing VCI, which will be found during the acknowledgment step. The switch then forwards the frame through port 3 to switch 2.
3. Switch 2 receives the setup request frame. The same events happen here as at switch 1; three columns of the table are completed: in this case, incoming port (l), incoming VCI (66), and outgoing port (2).
4. Switch 3 receives the setup request frame. Again, three columns are completed: incoming port (2), incoming VCI (22), and outgoing port (3).
5. Destination B receives the setup frame, and if it is ready to receive frames from A, it assigns a VCI to the incoming frames that come from A, in this case 77. This VCI lets the destination know that the frames come from A, and not other sources.

Acknowledgment: A special frame, called the acknowledgment frame, completes the entries in the switching tables.



Setup acknowledgment in a virtual-circuit network

1. The destination sends an acknowledgment to switch 3. The acknowledgment carries the global source and destination addresses so the switch knows which entry in the table is to be completed. The frame also carries VCI 77, chosen by the destination as the incoming VCI for frames from A. Switch 3 uses this VCI to complete the outgoing VCI column for this entry. Note that 77 is the incoming VCI for destination B, but the outgoing VCI for switch 3.
2. Switch 3 sends an acknowledgment to switch 2 that contains its incoming VCI in the table, chosen in the previous step. Switch 2 uses this as the outgoing VCI in the table.
3. Switch 2 sends an acknowledgment to switch 1 that contains its incoming VCI in the table, chosen in the previous step. Switch 1 uses this as the outgoing VCI in the table.
4. Finally switch 1 sends an acknowledgment to source A that contains its incoming VCI in the table, chosen in the previous step.
5. The source uses this as the outgoing VCI for the data frames to be sent to destination B.

Teardown Phase

In this phase, source A, after sending all frames to B, sends a special frame called a *teardown request.* Destination B responds with a teardown confirmation frame. All switches delete the corresponding entry from their tables.

Efficiency

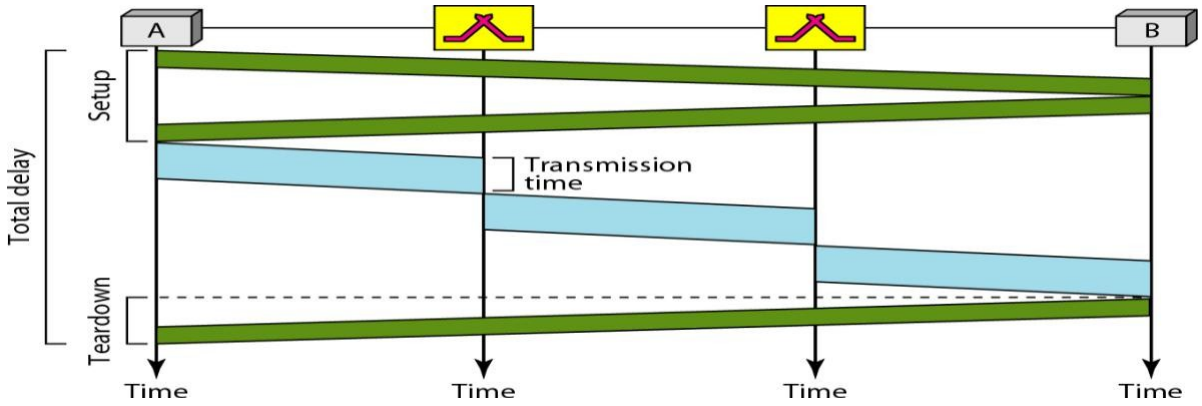
Resource reservation in a virtual-circuit network can be made during the setup or can be on demand during the data transfer phase.

In the first case, the delay for each packet is the same; in the second case, each packet may encounter different delays. There is one big advantage in a virtual-circuit network even if resource allocation is on demand. The source can check the availability of the resources, without actually reserving it.

Delay in Virtual-Circuit Networks

In a virtual-circuit network, there is a one-time delay for setup and a one-time delay for teardown.

If resources are allocated during the setup phase, there is no wait time for individual packets. Below Figure shows the delay for a packet traveling through two switches in a virtual-circuit network.



The packet is traveling through two switches (routers). There are three transmission times (3T), three propagation times (3τ), data transfer depicted by the sloping lines, a setup delay (which includes transmission and propagation in two directions), and a teardown delay (which includes transmission and propagation in one direction). We ignore the processing time in each switch. The total delay time is

$$\text{Total delay} = 3T + 3\tau + \text{setup delay} + \text{teardown delay}$$

Data can be corrupted during transmission. Some applications require that errors be detected and corrected.

Error Dection and Correction

Introduction

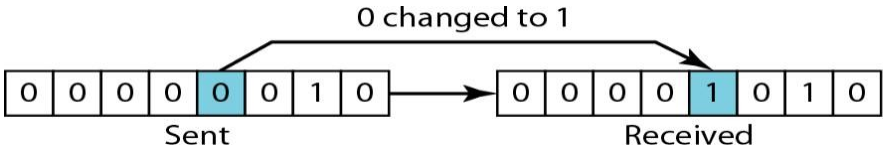
For many applications system must guarentee that the data received are same to the data transmitted.During transmission data may be corrupted because of many factors.Hence there shold be some mechanism to detect and correct errors.

Types of Errors

There are two types of error: Single bit error and Burst error.

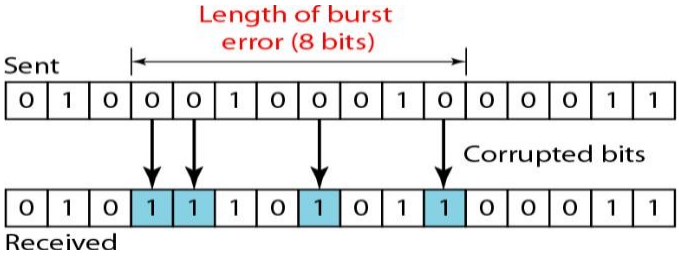
Single-Bit Error

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. Single-bit errors are the least likely type of error in serial data transmission.



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. In the below figure, 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.

Redundancy

To detect or correct errors some extra bits are sent with 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.

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.
- 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.

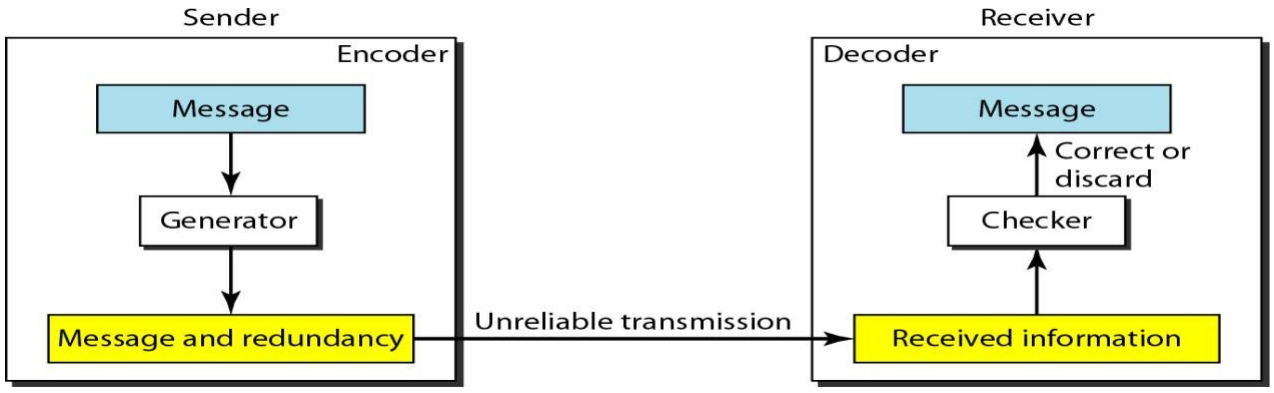
Forward Error Correction versus Retransmission

There are two main methods of error correction.

- **Forward error correction** is the process in which the receiver tries to guess the message by using redundant bits. This is possible, if the number of errors is small.
- **Correction by retransmission** is a technique in which the receiver detects the occurrence of an error and asks the sender to resend the message. Resending is repeated until a message arrives that the receiver believes is error-free.

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.
- Coding schemes can be divided into two broad categories: block coding and convolution coding.



Modular Arithmetic

In modular arithmetic only integers in the range 0 to N-1 is used. This is known as *modulo-N* arithmetic.

For example, if the modulus is 12, we use only the integers 0 to 11, inclusive.

Modulo-2 Arithmetic

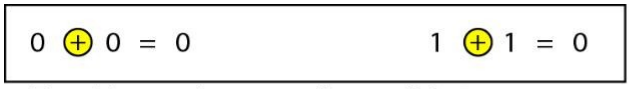
In this arithmetic, the modulus *N* is 2. We can use only 0 and 1. Operations in this arithmetic are very simple. The following shows how we can add or subtract 2 bits. Adding:

0+0=0 0+1=1 1+0=1 1+1=0

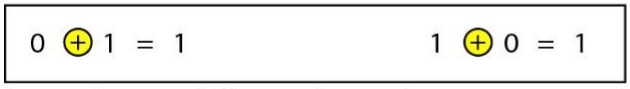
Subtracting:

0-0=0 0-1=1 1-0=1 1-1=0

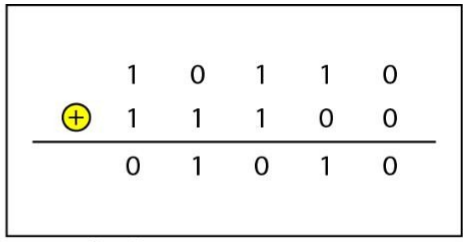
In this arithmetic we use the XOR (exclusive OR) operation for both addition and subtraction. The result of an XOR operation is 0 if two bits are the same; the result is I if two bits are different.



a. Two bits are the same, the result is 0.



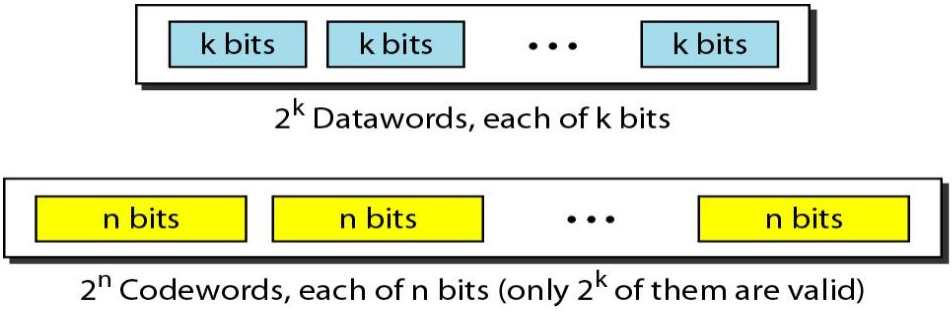
b. Two bits are different, the result is 1.



c. Result of XORing two patterns

3.1 Block Coding

- In block coding message is divided into k bits blocks called **datawords**. Then *r* redundant bits are added to each block to make the length $n = k + r$. The resulting *n-bit* blocks are called **codewords**.
- With k bits, we can create a combination of 2k datawords; with n bits, we can create a combination of 2n codewords.
- Since n > k, the number of possible codewords is larger than the number of possible datawords.
- The block coding process is one-to-one; the same dataword is always encoded as the same codeword. This means that we have 2n - 2k codewords that are not used. We call these codewords invalid or illegal.

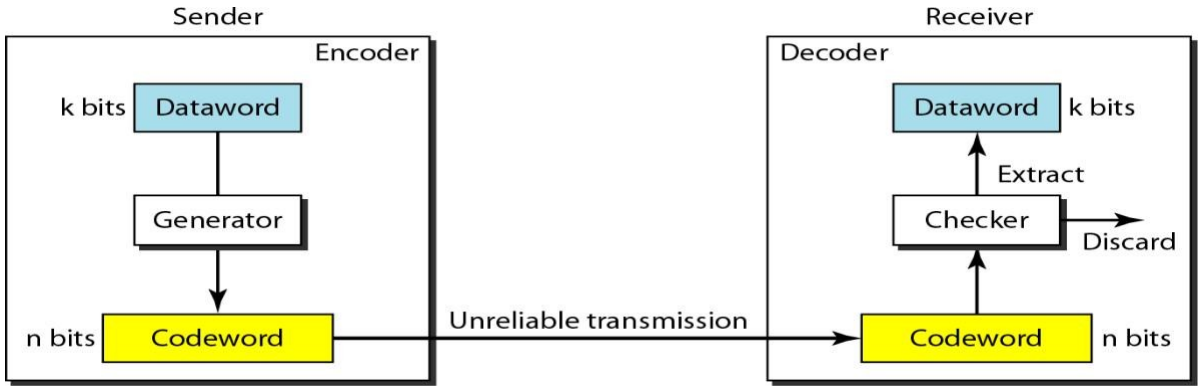


Error Detection

If the following two conditions are met, the receiver can detect a change in the original codeword.

2. The receiver has (or can find) a list of valid codewords.

3. The original codeword has changed to an invalid one.



Process of error detection in block coding

- The sender creates codewords out of datawords by using a generator that applies the rules and procedures of encoding.
- 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. Below Table shows the list of datawords and codewords.

Dataword	Codeword
00	000
01	011
10	101
11	110

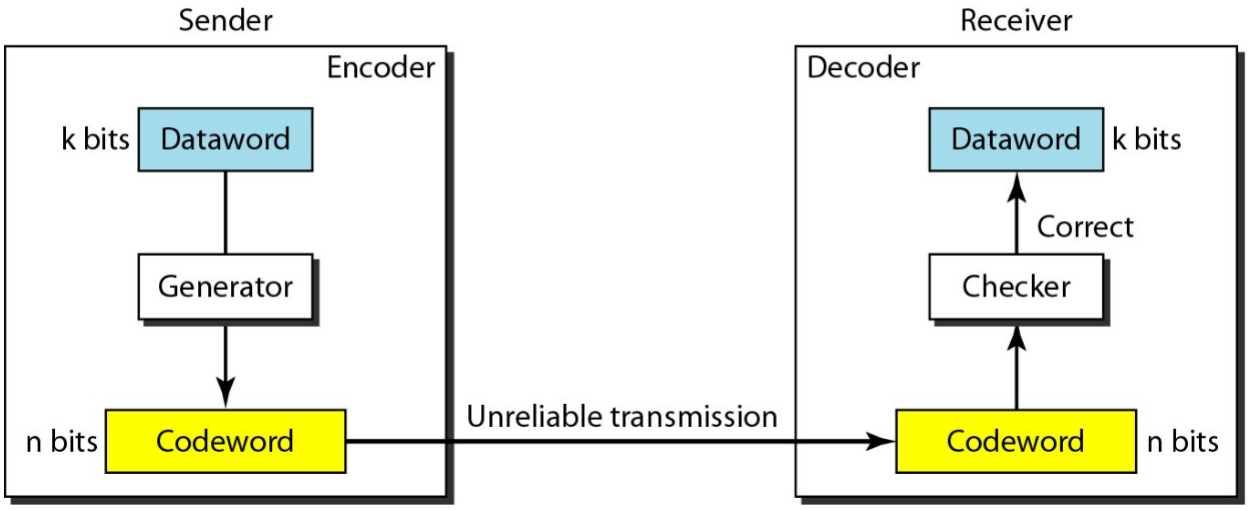
Assume the sender encodes the dataword 01 as 011 and sends it to the receiver. Consider the following cases:

3. The receiver receives 011. It is a valid codeword. The receiver extracts the dataword 01 from it.
4. The codeword is corrupted during transmission, and 111 is received (the leftmost bit is corrupted).

   This is not a valid codeword and is discarded.
5. 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

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.



Below Table shows the datawords and codewords.

Dataword	Codeword
00	00000
01	01011
10	10101
11	11110

Assume the dataword is 01. The sender consults the table (or uses an algorithm) to create the codeword 01011.

The codeword is corrupted during transmission, and 01001 is received (error in the second bit from the right).

First, the receiver finds that the received codeword is not in the table. This means an error has occurred. (Detection must come before correction.)

The receiver, assuming that there is only 1 bit corrupted, uses the following strategy to guess the correct dataword.

3. Comparing the received codeword with the first codeword in the table (01001 versus 00000), the receiver decides that the first codeword is not the one that was sent because there are two different bits.
4. By the same reasoning, the original codeword cannot be the third or fourth one in the table.
5. The original codeword must be the second one in the table because this is the only one that differs from the received codeword by 1 bit. The receiver replaces 01001 with 01011 and consults the table to find the dataword 01.

Hamming Distance

The Hamming distance between two words (of the same size) is the number of differences between the corresponding bits. Hamming distance between two words x and y is represented as d(x, y).The Hamming distance can be found by applying the XOR operation on the two words and counting the number of 1s in the result.

Example:

4. The Hamming distance d(000, 011) is 2 because 000 $\oplus$ 011 is 011 (two 1s).

5. The Hamming distance d(10101, 11110) is 3 because 10101 $\oplus$ 11110 is 01011 (three 1s).

Minimum Hamming Distance: The minimum Hamming distance is the smallest Hamming distance between all possible pairs in a set of words. It is represented as d_{min}.

Example 1

Find the minimum Hamming distance of the coding scheme in below table:

Dataword	Codeword
00	000
01	011
10	101
11	110

Solution

d(000,011) = 2, d(000,101) = 2, d(000,110) = 2, d(011,101) = 2, d(011,110) = 2, d(101,110) = 2 The $\mathbf{d_{min}}$ in this case is 2.

Example 2

Find the minimum Hamming distance of the coding scheme in below table:

Dataword	Codeword
00	00000
01	01011
10	10101
11	11110

Solution

d(00000,01011) = 3, d(00000,10101) = 3, d(00000,11110) = 4

d(01011,10101) = 4, d(01011,11110) = 3, d(10101,11110) =

3 The $\mathbf{d_{min}}$ in this case is 3.

Coding scheme needs to have at least three parameters: the codeword size *n,* the dataword size *k,* and the minimum Hamming distance *dmin.* A coding scheme C is written as *C(n, k)* with a separate expression for *dmin.*

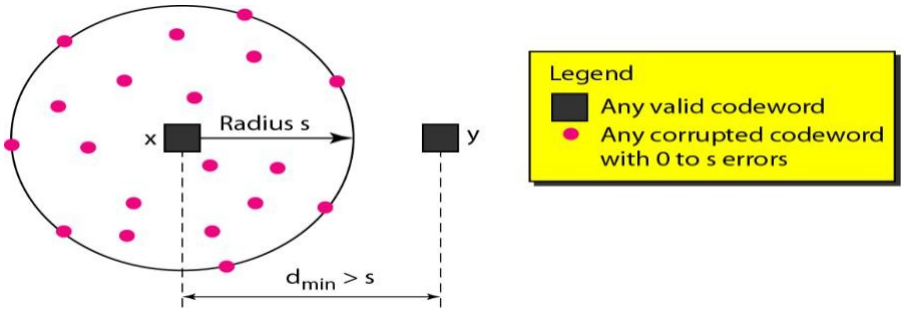
Ex: *C(5,* 2) with dmin = 3.

Hamming Distance and Error

- When a codeword is corrupted during transmission, the Hamming distance between the sent and received codewords is the number of bits affected by the error.
- The Hamming distance between the received codeword and the sent codeword is the number of bits that are corrupted during transmission.
- For example, if the codeword 00000 is sent and 01101 is received, 3 bits are in error and the Hamming distance between the two is d(00000, 01101) =3.

Minimum Distance for Error Detection

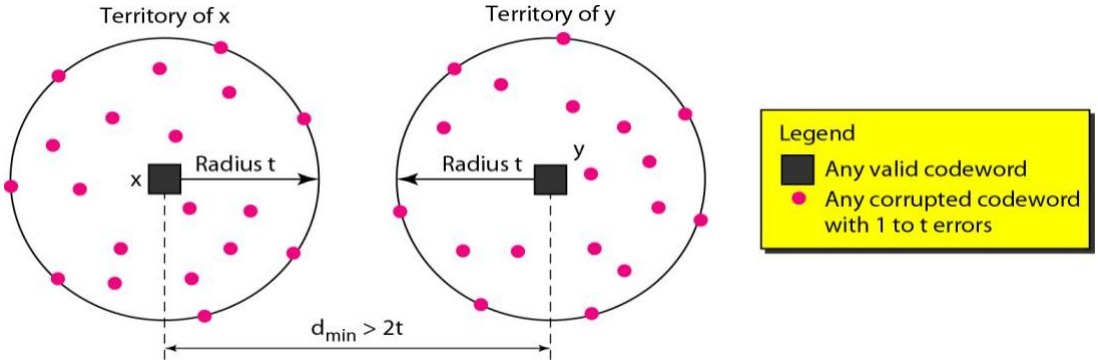
- If S errors occur during transmission, the Hamming distance between the sent codeword and received codeword is S.
- To guarantee the detection of up to S errors in all cases, the minimum Hamming distance in a block code must be dmin =*S* + 1.
- Let us assume that the sent codeword *x* is at the center of a circle with radius *S.* All other received codewords that are created by 1 to *S* errors are points inside the circle or on the perimeter of the circle. All other valid codewords must be outside the circle.



Minimum Distance for Error Correction

- When a received codeword is not a valid codeword, the receiver needs to decide which valid codeword was actually sent. The decision is based on the concept of territory, an exclusive area surrounding the codeword. Each valid codeword has its own territory.

- We use a geometric approach to define each territory. We assume that each valid codeword has a circular territory with a radius of *t* and that the valid codeword is at the center.
- For example, suppose a codeword *x* is corrupted by *t* bits or less. Then this corrupted codeword is located either inside or on the perimeter of this circle. If the receiver receives a codeword that belongs to this territory, it decides that the original codeword is the one at the center.
- To guarantee correction of up to t errors in all cases, the minimum Hamming distance in a block code must be dmin == 2t + 1.



3.2 Linear Block Codes

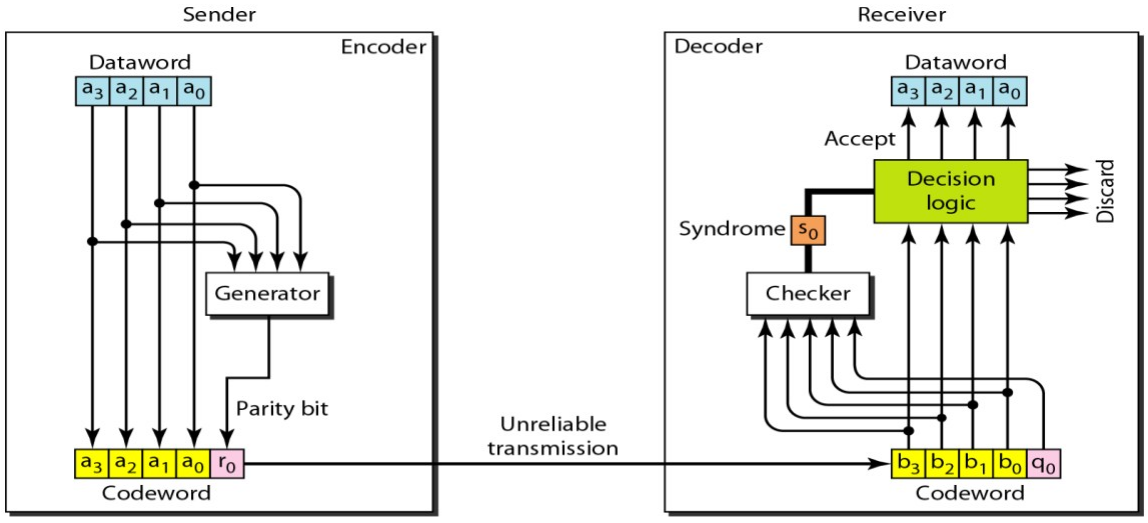
Linear block code is a code in which the exclusive OR of two valid codewords creates another valid codeword.

Minimum Distance for Linear Block Codes: The minimum Hamming distance is the number of 1s in the nonzero valid codeword with the smallest number of 1s.

Some Linear Block Codes

1) Simple Parity-Check Code

- In this code, a *k-bit* dataword is changed to an n-bit codeword where $n = k + 1$. The extra bit, called the parity bit, is selected to make the total number of 1s in the codeword even.
- A simple parity-check code is a single-bit error-detecting code in which $n = k + 1$ with dmin =2.



- The encoder uses a generator that takes a copy of a 4-bit dataword (a0, a1, a2, and a3) and generates a parity bit r_0.
- The dataword bits and the parity bit create the 5-bit codeword. The parity bit that is added makes the number of 1s in the codeword even.

  Example: Simple parity-check code C(5, 4)

Datawords	*Codewords*	*Datawords*	*Codewords*
0000	00000	1000	10001
0001	00011	1001	10010
0010	00101	1010	10100
0011	00110	1011	10111
0100	01001	1100	11000
0101	01010	1101	11011
0110	01100	1110	11101
0111	01111	1111	11110

- This is normally done by adding the 4 bits of the dataword (modulo-2); the result is the parity bit. In other words,

$$r_0 = a_3 + a_2 + a_1 + a_0 \text{ (modulo } - 2)$$

- If the number of 1s is even, the result is 0; if the number of 1s is odd, the result is 1. In both cases, the total number of 1s in the codeword is even.
- The sender sends the codeword which may be corrupted during transmission.
- The receiver receives a 5-bit word.
- The checker at the receiver does the same thing as the generator in the sender with one exception: The addition is done over all 5 bits.
- The result, which is called the syndrome, is just 1 bit. The syndrome is 0 when the number of 1s in the received codeword is even; otherwise, it is 1.

$$s_0 = b_3 + b_2 + b_1 + b_0 + q_0 \text{ (modulo – 2)}$$

- The syndrome is passed to the decision logic analyzer.
- If the syndrome is 0, there is no error in the received codeword; the data portion of the received codeword is accepted as the dataword.
- If the syndrome is 1, the data portion of the received codeword is discarded. The dataword is not created.

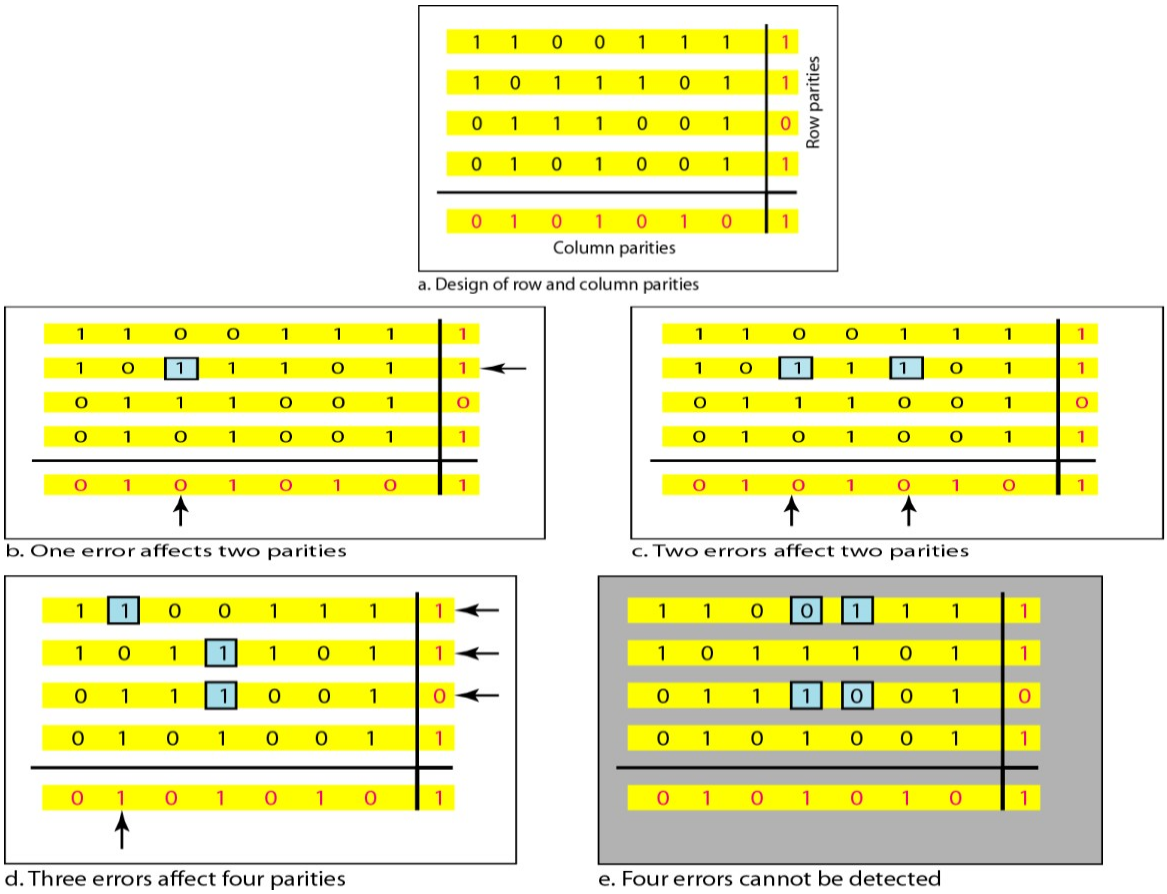
Assume the sender sends the dataword 1011. The codeword created from this dataword is 10111, which is sent to the receiver.

6. No error occurs; the received codeword is 10111. The syndrome is O. The dataword 1011 is created.
7. One single-bit error changes a_1 The received codeword is 10011. The syndrome is 1. No dataword is created.
8. One single-bit error changes r_0 The received codeword is 10110. The syndrome is 1. No dataword is created. Note that although none of the dataword bits are corrupted, no dataword is created because the code is not sophisticated enough to show the position of the corrupted bit.
9. An error changes *ro* and a second error changes a_3 The received codeword is 00110. The syndrome is 0. The dataword 0011 is created at the receiver. Note that here the dataword is wrongly created due to the syndrome value. The simple parity-check decoder cannot detect an even number of errors. The errors cancel each other out and give the syndrome a value of 0.

5. Three bits-a_3, a_2, and a_1 are changed by errors. The received codeword is 01011. The syndrome is 1. The dataword is not created. This shows that the simple parity check, guaranteed to detect one single error, can also find any odd number of errors.

Limitation: A simple parity-check code can detect an odd number of errors.

- A better approach is the two-dimensional parity check. In this method, the dataword is organized in a table.
- The data to be sent, five 7-bit bytes, are put in separate rows.
- For each row and each column, 1 parity-check bit is calculated.
- The whole table is then sent to the receiver, which finds the syndrome for each row and each column.
- The two-dimensional parity check can detect up to three errors that occur anywhere in the table. However, errors affecting 4 bits may not be detected.



a. Design of row and column parities

b. One error affects two parities

c. Two errors affect two parities

d. Three errors affect four parities

e. Four errors cannot be detected

Hamming Codes

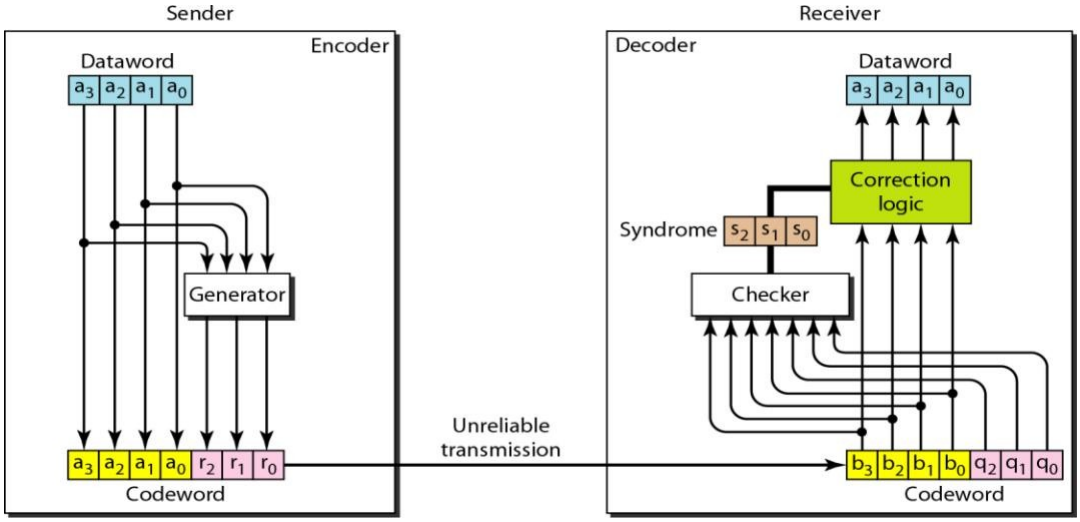
These codes were originally designed with *dmin* = 3, which means that they can detect up to two errors or correct one single error.

In hamming code we need to choose an integer m, say *m* >= 3. The values of n and *k* are then calculated from m as $n = 2m - 1$ and $k = n - m$. The number of check bits $r = m$. Eg: if m = 3, n=7, k = 4

Hamming code C(7, 4) - n=7, k = 4:

Datawords	*Codewords*	*Datawords*	*Codewords*
0000	0000000	1000	1000110
0001	0001101	1001	1001011
0010	0010111	1010	1010001
0011	0011010	1011	1011100
0100	0100011	1100	1100101
0101	0101110	1101	1101000
0110	0110100	1110	1110010
0111	0111001	1111	1111111

Below figure shows the structure of the encoder and decoder:



A copy of a 4-bit dataword is fed into the generator that creates three parity checks.

$\mathbf{r_0 = a_2 + a_1 + a_0}$ modulo-2

$\mathbf{r_1 = a_3 + a_2 + a_1}$ modulo-2

$\mathbf{r_2 = a_1 + a_0 + a_3}$ modulo-2

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:

$\mathbf{s_0 = b_2 + b_1 + b_0}$ modulo-2

$\mathbf{s_1 = b_3 + b_2 + b_1}$ modulo-2

$\mathbf{s_2 = b_1 + b_0 + b_3}$ modulo-2

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.

Syndrome	000	001	010	011	100	101	110	111
Error	None	q_0	q_1	b_2	q_2	b_0	b_3	b_1

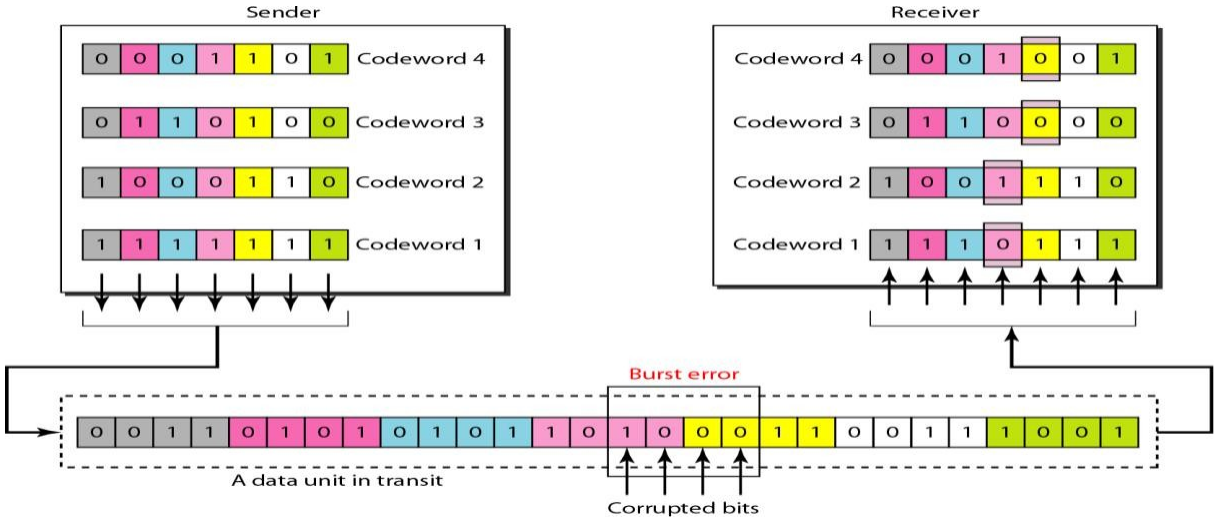
For example, if q_0 is in error, S_0 is the only bit affected; the syndrome, therefore, is 001. If b_2 is in error, S_0 and S_1 are the bits affected; the syndrome therefore is 011. Similarly, if b_1 is in error, all 3 syndrome bits are affected and the syndrome is 111.

Example:

1. The dataword 0100 becomes the codeword 0100011. The codeword 0100011 is received. The syndrome is 000 (no error), the final dataword is 0100.
2. The dataword 0111 becomes the codeword 0111001. The codeword 0011001 is received. The syndrome is 011. Therefore b_2 is in error. After flipping b_2 (changing the 1 to 0), the final dataword is 0111.
3. The dataword 1101 becomes the codeword 1101000. The codeword 0001000 is received (two errors). The syndrome is 101, which means that b_0 is in error. After flipping b_0, we get 0000, the wrong dataword. This shows that our code cannot correct two errors.

Performance

A Hamming code can only correct a single error or detect a double error. However, there is a way to make it detect a burst error.



The key is to split a burst error between several codewords, one error for each codeword.

To make the Hamming code respond to a burst error of size *N,* we need to make *N* codewords out of our frame. Then, instead of sending one codeword at a time, we arrange the codewords in a table and send the bits in the table a column at a time.

In the above Figure, the bits are sent column by column (from the left). In each column, the bits are sent from the bottom to the top. In this way, a frame is made out of the four codewords and sent to the receiver. It is shown in the figure that when a burst error of size 4 corrupts the frame, only 1 bit from each codeword is corrupted. The corrupted bit in each codeword can then easily be corrected at the receiver.

3.3 Cyclic Codes

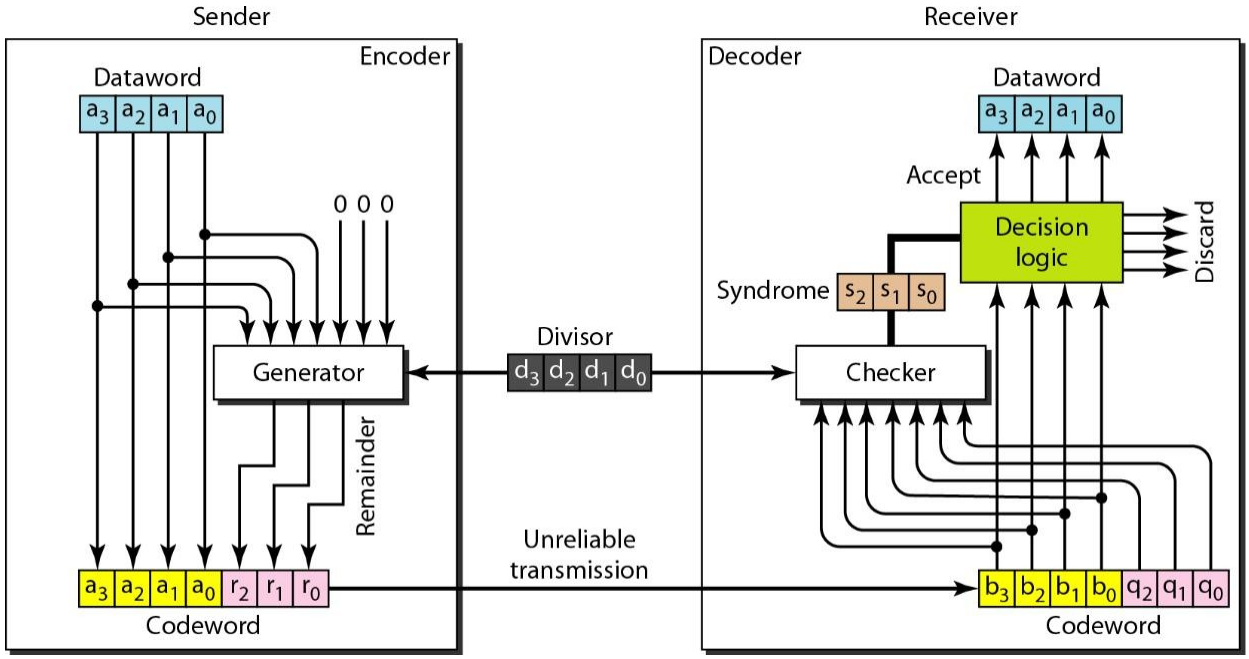
Cyclic codes are special linear block codes in which, if a codeword is cyclically shifted (rotated), the result is another codeword.

For example, if 1011000 is a codeword and we cyclically left-shift, then 0110001 is also a codeword.

In this case, if we call the bits in the first word a_0 to a_6 and the bits in the second word b_0 to b_6, we can shift the bits by using the following:

$b_1 = a_0$ $b_2 = a_1$ $b_3 = a_2$ $b_4 = a_3$ $b_5 = a_4$ $b_6 = a_5$ $b_0 = a_6$

Cyclic Redundancy Check



Below Table shows an example of a CRC code.

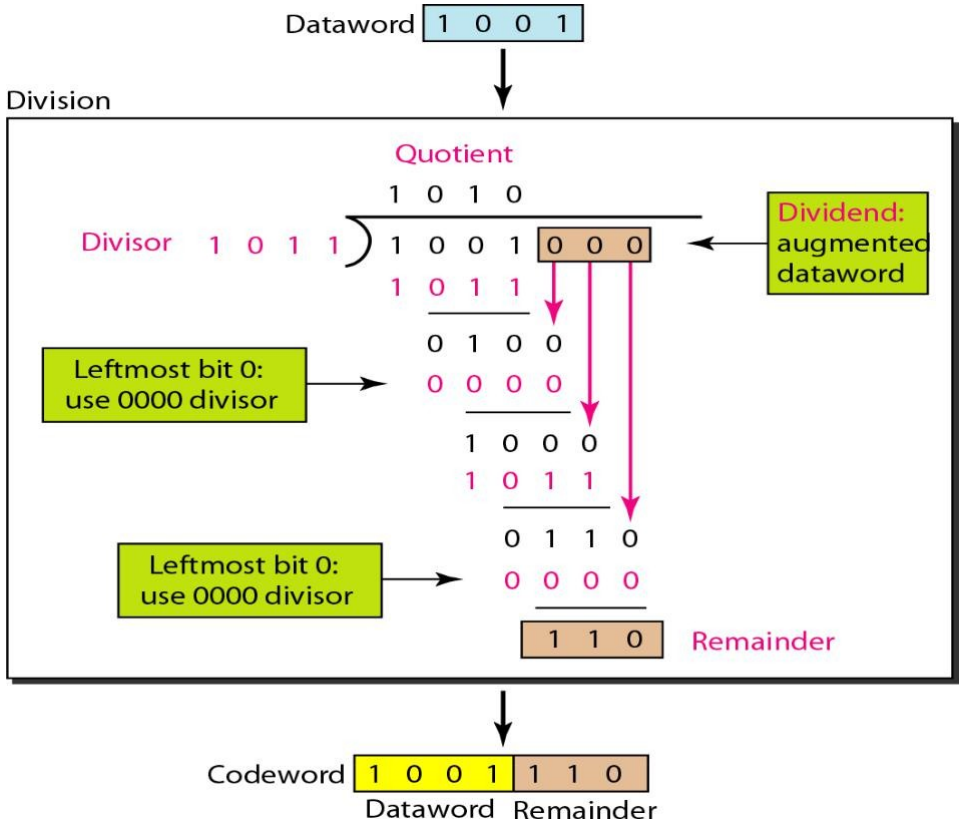
Dataword	*Codeword*	*Dataword*	*Codeword*
0000	0000000	1000	1000101
0001	0001011	1001	1001110
0010	0010110	1010	1010011
0011	0011101	1011	1011000
0100	0100111	1100	1100010
0101	0101100	1101	1101001
0110	0110001	1110	1110100
0111	0111010	1111	1111111

- In the encoder, the dataword has k bits (4 here); the codeword has n bits (7 here). The size of the dataword is augmented by adding n - k (3 here) 0s to the right-hand side of the word. The n-bit result is fed into the generator.

- The generator uses a divisor of size n - k + 1 (4 here), predefined and agreed upon. The generator divides the augmented dataword by the divisor (modulo-2 division).
- The quotient of the division is discarded; the remainder is appended to the dataword to create the codeword.
- The decoder receives the possibly corrupted codeword. A copy of all n bits is fed to the checker which is a replica of the generator.
- The remainder produced by the checker is a syndrome of *n* - *k* (3 here) bits, which is fed to the decision logic analyzer.
- The analyzer has a simple function. If the syndrome bits are all as, the 4 leftmost bits of the codeword are accepted as the dataword (interpreted as no error); otherwise, the 4 bits are discarded (error).

Encoder

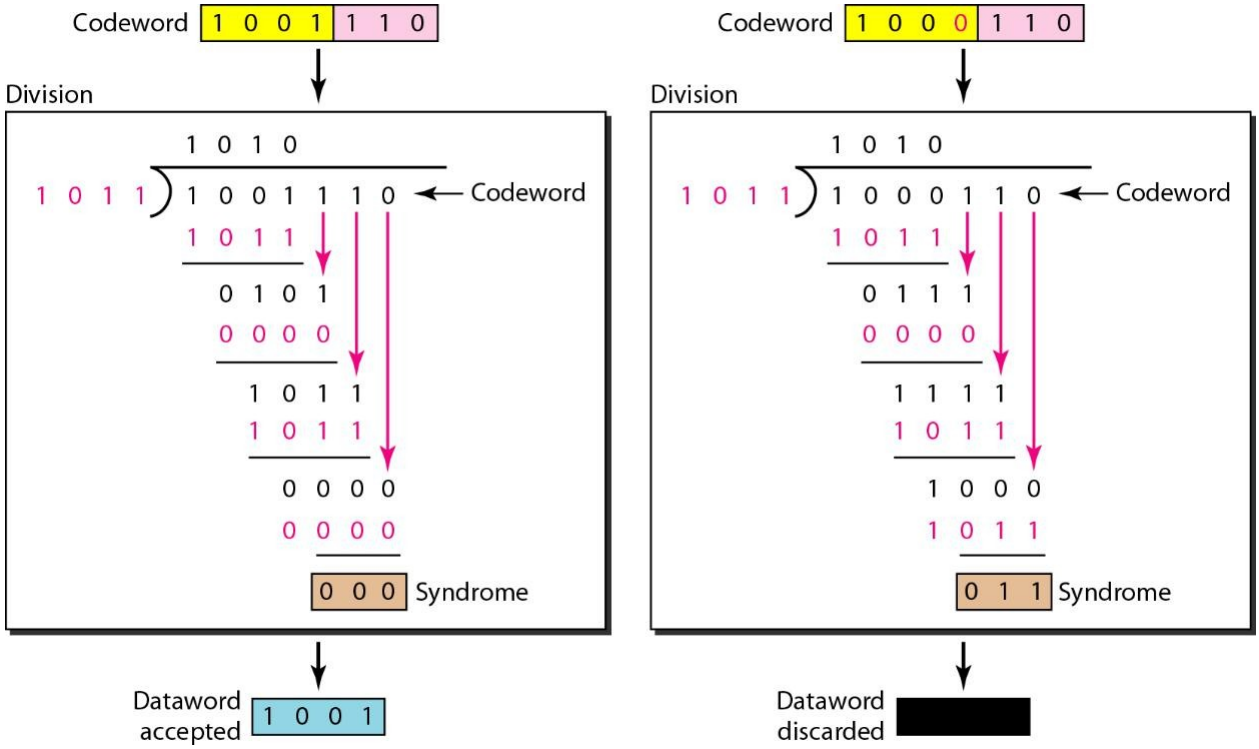
The encoder takes the dataword and augments it with n - k number of 0s. It then divides the augmented dataword by the divisor.



Decoder

The codeword can change during transmission. The decoder does the same division process as the encoder. The remainder of the division is the syndrome. If the syndrome is all 0s, there is no error; the dataword is separated from the received codeword and accepted. Otherwise, everything is discarded.

The left hand figure shows the value of syndrome when no error has occurred; the syndrome is 0. The right-hand part of the figure shows the case in which there is one single error. The syndrome is not all 0s



Hardware Implementation

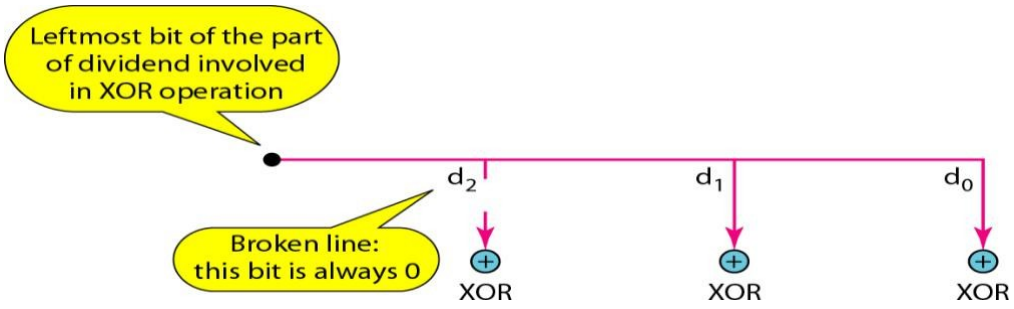
One of the advantages of a cyclic code is that the encoder and decoder can easily and cheaply be implemented in hardware by using a handful of electronic devices. Also, a hardware implementation increases the rate of check bit and syndrome bit calculation.

Divisor:

1. The divisor is repeatedly XORed with part of the dividend.
2. The divisor has $n - k + 1$ bits which either are predefined or are all Os. In other words, the bits do not change from one dataword to another. In previous example, the divisor bits were

either 1011 or 0000. The choice was based on the leftmost bit of the part of the augmented data bits that are active in the XOR operation.

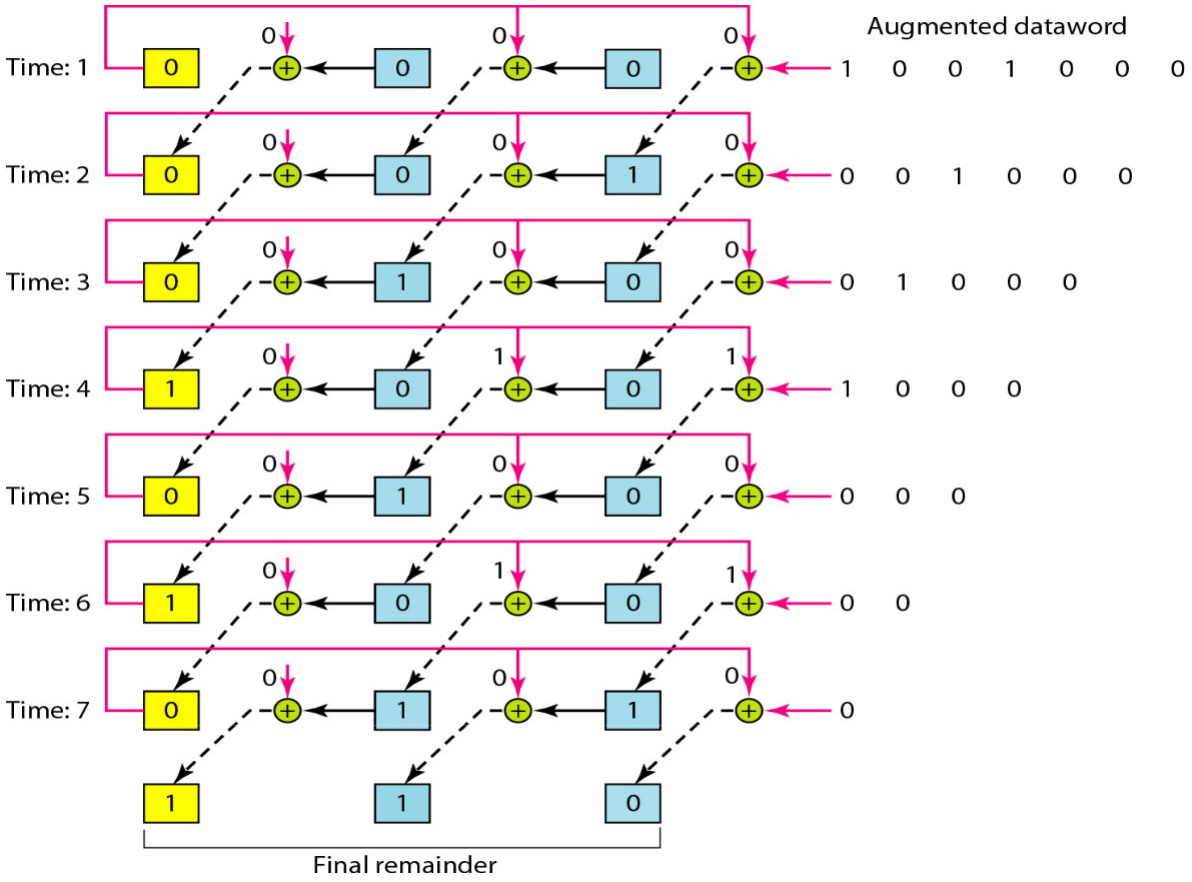
3. A close look shows that only *n* - *k* bits of the divisor is needed in the XOR operation. The leftmost bit is not needed because the result of the operation is always 0, no matter what the value of this bit. The reason is that the inputs to this XOR operation are either both 0s or both 1s.



Steps:

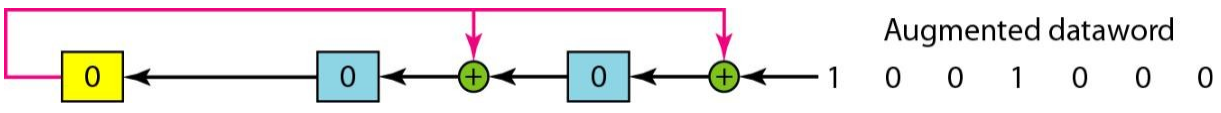
1. Assume that the remainder is originally all Os (000 in our example).
2. At each time click (arrival of 1 bit from an augmented dataword), repeat the following two actions:
   a. Use the leftmost bit to make a decision about the divisor (011 or 000).
   b. The other 2 bits of the remainder and the next bit from the augmented dataword (total of 3 bits) are XORed with the 3-bit divisor to create the next remainder.

Below Figure shows this simulator, but note that this is not the final design; there will be more improvements.

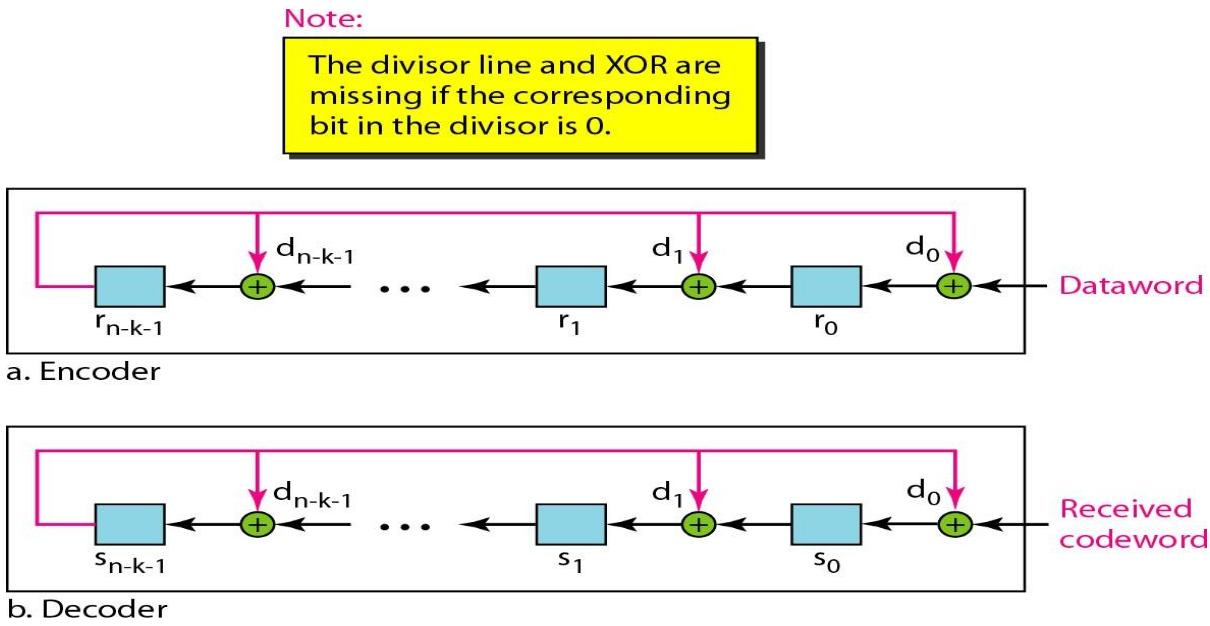


At each clock tick, shown as different times, one of the bits from the augmented dataword is used in the XOR process.

The above design is for demonstration purposes only. It needs simplification to be practical. First, we do not need to keep the intermediate values of the remainder bits; we need only the final bits. We therefore need only 3 registers instead of 24. After the XOR operations, we do not need the bit values of the previous remainder. Also, we do not need 21 XOR devices; two are enough because the output of an XOR operation in which one of the bits is 0 is simply the value of the other bit. This other bit can be used as the output. With these two modifications, the design becomes tremendously simpler and less expensive, as shown below

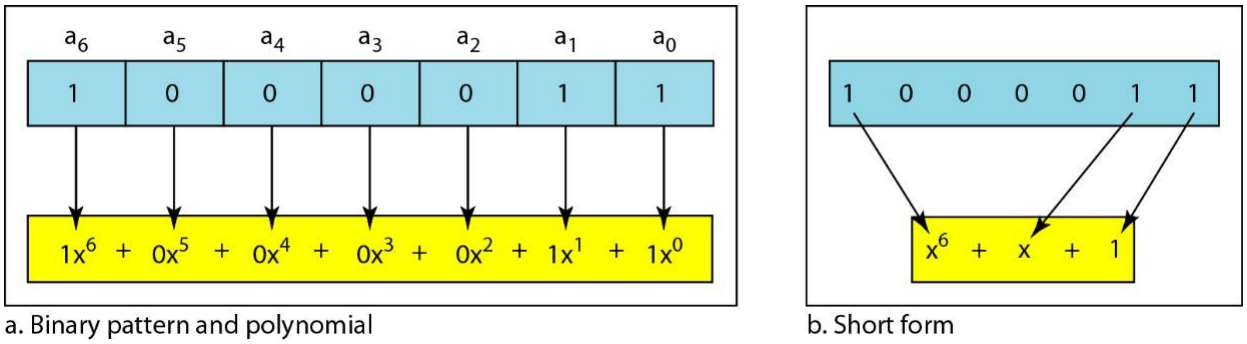


General Design



Polynomials

A pattern of Os and 1s can be represented as a **polynomial** with coefficients of 0 and 1. The power of each term shows the position of the bit; the coefficient shows the value of the bit. Figure shows a binary pattern and its polynomial representation.



Degree of a Polynomial

The degree of a polynomial is the highest power in the polynomial.

For example, the degree of the polynomial $x6 + x + 1$ is 6. Note that the degree of a polynomial is 1 less that the number of bits in the pattern. The bit pattern in this case has 7 bits.

Adding and Subtracting Polynomials

Adding and subtracting polynomials in mathematics are done by adding or subtracting the coefficients of terms with the same power. In our case, the coefficients are only 0 and 1, and adding is in modulo-2. This has two consequences. First, addition and subtraction are the same. Second, adding or subtracting is done by combining terms and deleting pairs of identical terms. For example, adding *x5 + x4 + x2* and *x6 + x4 + x2* gives just *x6 + x5.* The terms *x4* and *x2* are deleted. However, note that if we add, for example, three polynomials and we get *x2* three times, we delete a pair of them and keep the third.

Multiplying or Dividing Terms

In this arithmetic, multiplying a term by another term is very simple; we just add the powers.
For example, x3 x x4 is x7 , For dividing, we just subtract the power of the second term from the power of the first.

Multiplying Two Polynomials

Multiplying a polynomial by another is done term by term. Each term of the first polynomial must be multiplied by all terms of the second. The result, of course, is then simplified, and pairs of equal terms are deleted. The following is an example:

$$(x^5 + x^3 + x^2 + x)(x^2 + x + 1)$$
$$= x^7 + x^6 + x^5 + x^5 + x^4 + x^3 + x^4 + x^3 + x^2 + x^3 + x^2 + x$$
$$= x^7 + x^6 + x^3 + x$$

Dividing One Polynomial by Another

We divide the first term of the dividend by the first term of the divisor to get the first term of the quotient. We multiply the term in the quotient by the divisor and subtract the result from the dividend. We repeat the process until the dividend degree is less than the divisor degree.

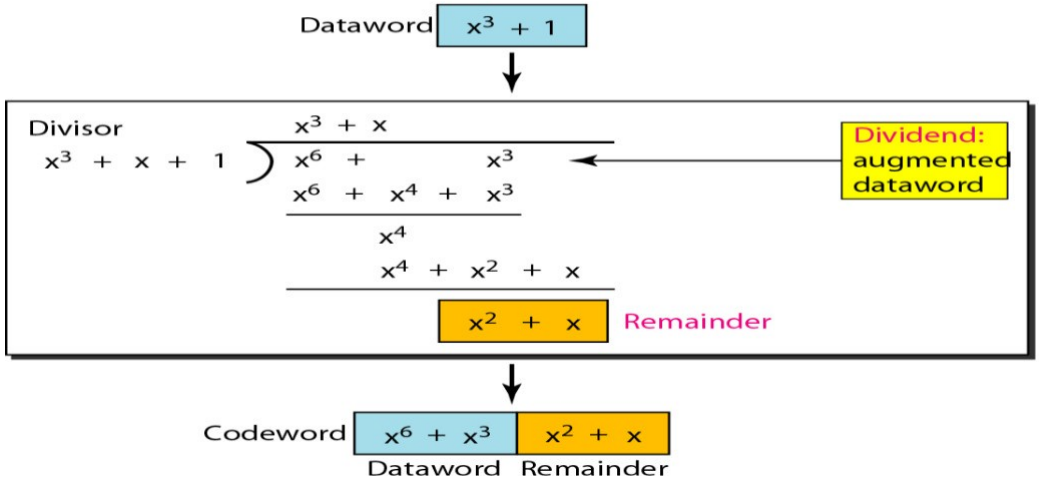
Shifting

A binary pattern is often shifted a number of bits to the right or left. Shifting to the left means adding extra Os as rightmost bits; shifting to the right means deleting some rightmost bits. Shifting to the left is accomplished by multiplying each term of the polynomial by xn, where m

is the number of shifted bits; shifting to the right is accomplished by dividing each term of the polynomial by x^n. The following shows shifting to the left and to the right. Note that we do not have negative powers in the polynomial representation.

Shifting left 3 bits: 10011 becomes 10011000 $x^4 + x + 1$ becomes $x^7 + x^4 + x^3$

Shifting right 3 bits: 10011 becomes 10 $x^4 + x + 1$ becomes x



Cyclic Code Analysis

Following notations can be used in the cyclic codes:

Dataword: *d(x)* Syndrome: *s(x)* Codeword: *c(x)*

Error: *e(x)* Generator: *g(x)*

In a cyclic code,

1. If s(x) != 0, one or more bits is corrupted.
2. If s(x) = 0, either
   a. No bit is corrupted. or
   b. Some bits are corrupted, but the decoder failed to detect them.

The received codeword is the sum of the sent codeword and the error.

Received codeword =*c(x)* + *e(x)*

The receiver divides the received codeword by g(x) to get the syndrome.

$$\frac{\text{Received codeword}}{g(x)} = \frac{c(x)}{g(x)} + \frac{e(x)}{g(x)}$$

The Right hand side of above equation is called as syndrome.

If Syndrome does not have a remainder (syndrome =0), either *e(x)* is 0 or *e(x)* is divisible by *g(x)*.

In a cyclic code, those *e(x)* errors that are divisible by *g(x)* are not caught.

Single-Bit Error

A single-bit error is $e(x) = x^i$, where *i* is the position of the bit. If a single-bit error is caught, then x^i is not divisible by *g(x)*.

If the generator has more than one term and the coefficient of x0 is 1, all single errors can be caught.

Example:

Which of the following *g(x)* values guarantees that a single-bit error is caught? For each case, what is the error that cannot be caught?

a. $x + 1$

b. *x3*

c. 1

Solution:

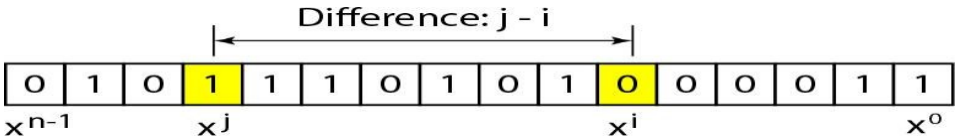
a. No x^i can be divisible by $x + 1$. In other words, $x^i / (x + 1)$ always has a remainder. So the syndrome is nonzero. Any single-bit error can be caught.

b. If i is equal to or greater than 3, x^i is divisible by *g(x)*. The remainder of x^i / x^3 is zero, and the receiver is fooled into believing that there is no error, although there might be one. All single-bit errors in positions 1 to 3 are caught.

c. All values of *i* make *j* divisible by *g(x)*. No single-bit error can be caught

Two Isolated Single-Bit Errors

Two isolated single bit errors can be represented as $e(x) = x^j + x^i$. The values of i and j define the positions of the errors, and the difference j - i defines the distance between the two errors.



We can write $e(x) = x^i (x^{j-i} + 1) = x^j (x^t + 1)$.

If a generator cannot divide x be detected.

+ 1 (*i* between 0 and *n* – 1), then all isolated double errors can

Example:

Find the status of the following generators related to two isolated, single-bit errors.

a. $x + 1$

b. $x^4 + 1$

c. $x^7 + x^6 + 1$

$x^{15} + x^d$

$14 + 1$

Solution:

a. This is a very poor choice for a generator. Any two errors next to each other cannot be detected.

b. This generator cannot detect two errors that are four positions apart. The two errors can be anywhere, but if their distance is 4, they remain undetected.

c. This is a good choice for this purpose.

d. This polynomial cannot divide any error of *type* $x^t + 1$ if t is less than 32,768. This means that a codeword with two isolated errors that are next to each other or up to 32,768 bits apart can be detected by this generator.

Odd Numbers of Errors

A generator that contains a factor of x + 1 can detect all odd-numbered errors.

Burst Errors

If L is the burst size and r is the degree of generator polynomial.

- All burst errors with L <= r will be detected.
- All burst errors with L = r + 1 will be detected with probability $1 - (1/2)^{r-1}$
- All burst errors with L > r + 1 will be detected with probability $1- (1/2)^r$

Example:

Find the suitability of the following generators in relation to burst errors of different lengths.

a. x6 + 1

b. x18+ x7 + x + 1

c. x32+x23+x7+1

Solution:

a. This generator can detect all burst errors with a length less than or equal to 6 bits; 3 out of 100 burst errors with length 7 will slip by; 16 out of 1000 burst errors of length 8 or more will slip by.

b. This generator can detect all burst errors with a length less than or equal to 18 bits; 8 out of 1 million burst errors with length 19 will slip by; 4 out of I million burst errors of length 20 or more will slip by.

c. This generator can detect all burst errors with a length less than or equal to 32 bits; 5 out of 10 billion burst errors with length 33 will slip by; 3 out of 10 billion burst errors of length 34 or more will slip by.

A good polynomial generator needs to have the following characteristics:

1. It should have at least two terms.
2. The coefficient of the term x0 should be 1.
3. It should not divide $x^t + 1$, for t between 2 and $n - 1$.
4. It should have the factor $x + 1$.

Standard Polynomials

Name	*Polynomial*	*Application*
CRC-8	$x^8 + x^2 + x + 1$	ATM header
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$	ATM AAL
CRC-16	$x^{16} + x^{12} + x^5 + 1$	HDLC
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	LANs

Advantages of Cyclic Codes

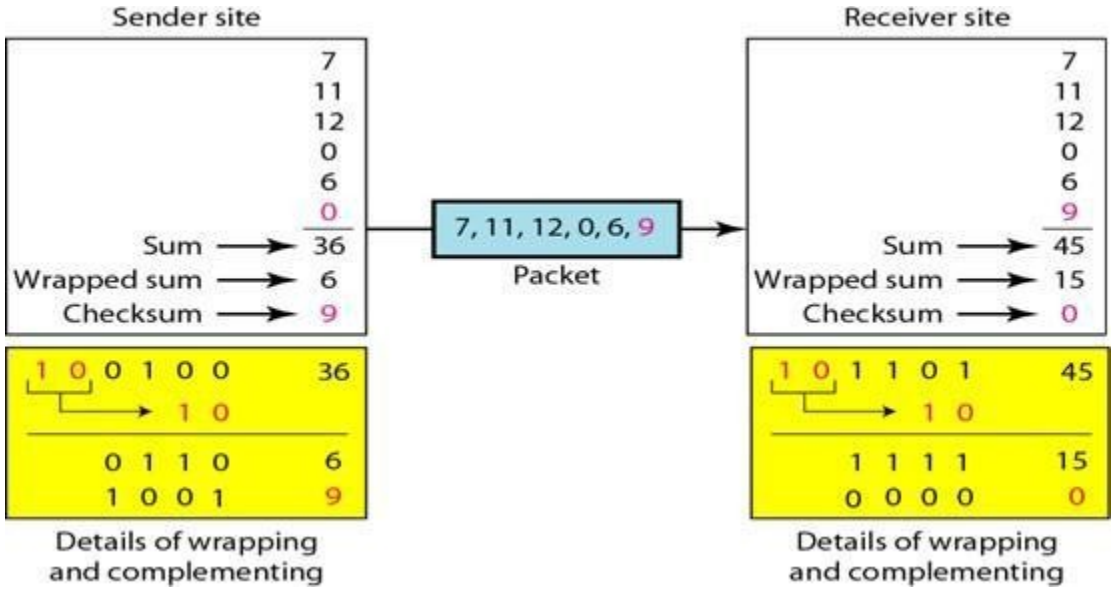
- Cyclic codes have a very good performance in detecting single-bit errors, double errors, an odd number of errors, and burst errors.

- They can easily be implemented in hardware and software.
- They are especially fast when implemented in hardware.

3.4 Checksum

The checksum is used in the Internet by several protocols. The checksum is based on the concept of redundancy.

Below Figure shows the process at the sender and at the receiver. The sender initializes the checksum to 0 and adds all data items and the checksum (the checksum is considered as one data item and is shown in color). The result is 36. However, 36 cannot be expressed in 4 bits. The extra two bits are wrapped and added with the sum to create the wrapped sum value 6. In the figure, we have shown the details in binary. The sum is then complemented, resulting in the checksum value 9 (15 - 6 = 9). The sender now sends six data items to the receiver including the checksum 9. The receiver follows the same procedure as the sender. It adds all data items (including the checksum); the result is 45. The sum is wrapped and becomes 15. The wrapped sum is complemented and becomes 0. Since the value of the checksum is 0, this means that the data is not corrupted. The receiver drops the checksum and keeps the other data items. If the checksum is not zero, the entire packet is dropped.



Internet Checksum

Traditionally, the Internet has been using a 16-bit checksum.

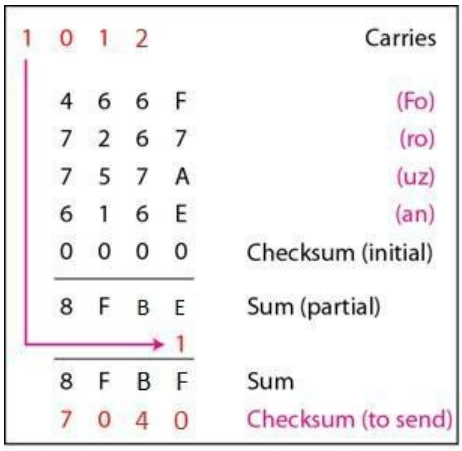
Sender site:

1. The message is divided into 16-bit words.
2. The value of the checksum word is set to 0.
3. All words including the checksum are added using one's complement addition.
4. The sum is complemented and becomes the checksum.
5. The checksum is sent with the data.

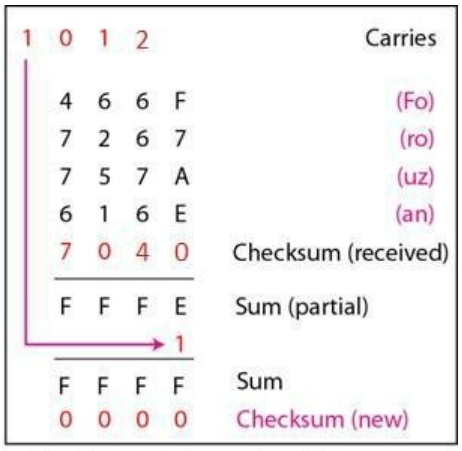
Receiver site:

1. The message (including checksum) is divided into 16-bit words.
2. All words are added using one's complement addition.
3. The sum is complemented and becomes the new checksum.
4. If the value of checksum is 0, the message is accepted; otherwise, it is rejected.

Example:



a. Checksum at the sender site



a. Checksum at the receiver site

Performance

The traditional checksum uses a small number of bits (16) to detect errors in a message of any size (sometimes thousands of bits). However, it is not as strong as the CRC in error-checking capability. For example, if the value of one word is incremented and the value of another word is decremented by the same amount, the two errors cannot be detected because the sum and checksum remain the same. Also if the values of several words are incremented but the total change is a multiple of 65535, the sum and the checksum does not change, which means the errors are not detected.