Line coding graphs questions not included.

Explain TCP/IP model with diagram.

TCP/IP Model Explained

The TCP/IP model is a hierarchical protocol suite that was developed before the OSI model. The TCP/IP model is named after its two most important protocols: the Transmission Control Protocol (TCP) and the Internet Protocol (IP). While the OSI model is a theoretical framework, the TCP/IP model is the actual model used in today's data communications. The TCP/IP model has five layers: physical, data link, network, transport, and application. The first four layers provide physical standards, network interfaces, internetworking, and transport functions that correspond to the first four layers of the OSI model. The three topmost layers in the OSI model, however, are represented in TCP/IP by a single layer called the application layer. Each upper-level protocol is supported by one or more lower-level protocols.

Here is a description of each layer in the TCP/IP model:

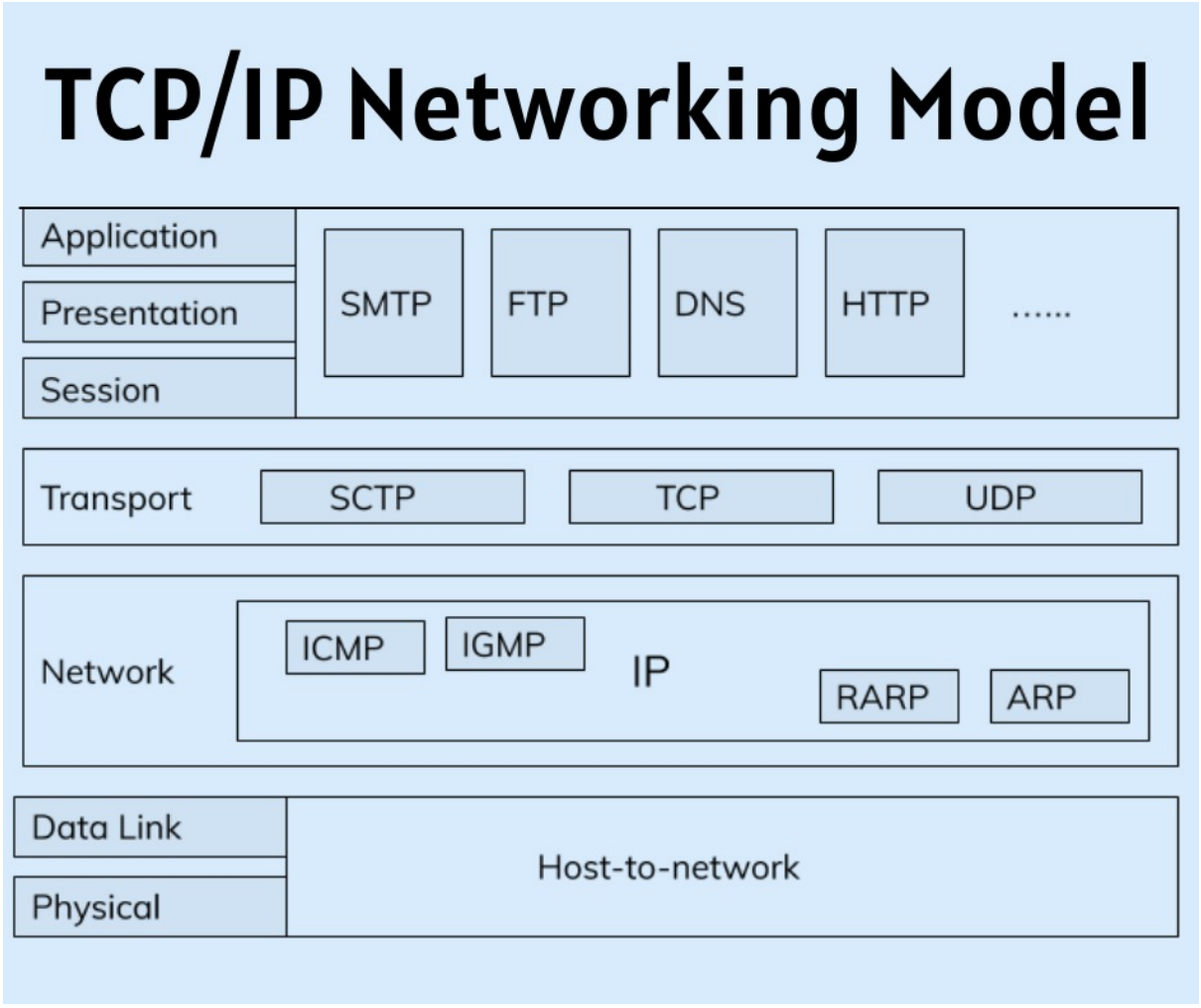
- **Application Layer:** This layer enables users to access the network. It provides services to the user. The application layer in TCP/IP combines the session, presentation, and application layers of the OSI model. Some protocols at this layer include:
    - HTTP (Hypertext Transfer Protocol), which provides for Web document request and transfer
    - SMTP (Simple Mail Transfer Protocol), which provides for the transfer of e-mail messages
    - FTP (File Transfer Protocol), which provides for the transfer of files between two end systems
    - DNS (Domain Name System), which translates human-friendly names for Internet end systems to a 32-bit network address
- **Transport Layer:** This layer is responsible for the delivery of a message from one process to another. It is responsible for process-to-process delivery. The transport layer also breaks long messages into shorter segments. The two main protocols at this layer are TCP and UDP:

    - **TCP** provides a connection-oriented service to its applications. This service includes guaranteed delivery of application-layer messages to the destination and flow control. TCP is a reliable stream transport protocol.
    - **UDP** provides a connectionless service to its applications. This is a no-frills service that provides no reliability, no flow control, and no congestion control. UDP is a connectionless, unreliable transport protocol that adds only port addresses, checksum error control, and length information to the data from the upper layer.
- **Network Layer:** This layer is responsible for host-to-host communication. It is responsible for the delivery of datagrams between two hosts. The network layer's main protocol in TCP/IP is the Internet Protocol (IP). The network layer also uses addresses called IP addresses to uniquely define a host on the Internet. IP provides logical communication between hosts. The IP service model is a best-effort delivery service, meaning that it does not guarantee segment delivery, orderly delivery of segments, or the integrity of the data in the segments. Some other protocols that support the network layer include:
    - ARP (Address Resolution Protocol), which finds the hardware address of a host from an IP address.
    - RARP (Reverse Address Resolution Protocol), which finds the IP address of a host from a hardware address.
    - ICMP (Internet Control Message Protocol), which handles error reporting and sends query messages.
    - IGMP (Internet Group Message Protocol), which facilitates the simultaneous transmission of a message to a group of recipients.
- **Data Link Layer:** This layer transforms the physical layer into a link responsible for node-to-node communication. The data link layer is responsible for node-to-node delivery. The data link layer adds a header to the frame to define the addresses of the sender and receiver of the frame. It also adds reliability to the physical layer by

adding mechanisms to detect and retransmit damaged or lost frames.

- **Physical Layer:** This layer coordinates the functions required to carry a bit stream over a physical medium. It deals with the mechanical and electrical specifications of the interface and transmission medium. It also defines the type of encoding, or how 0s and 1s are changed to signals. The physical layer also defines the transmission rate.

The TCP/IP model does not define any specific protocol at the physical and data link layers. It supports all the standard and proprietary protocols. A network in a TCP/IP internetwork can be a LAN or a WAN.



Define socket. List all the socket functions for TCP server and client side.

Socket Definition and Functions

A **socket** is the interface between the application layer and the transport layer within a host. It is also referred to as the Application Programming Interface (API) between the application and the network, since the socket is the programming interface with which network applications are built. The application developer has control of everything on the application-layer side of the socket but has little control of the transport-layer side of the socket.

The combination of an **IP address** and a **port number** is called a **socket address**. The client socket address defines the client process uniquely, just as the server socket address defines the server process uniquely.

- The **IP address** selects the **host**.
- The **port number** selects the **process**.

The sources provide information on socket functions for TCP (Transmission Control Protocol) connections, but not a complete list of all functions.

TCP Socket Functions: Server Side

On the server side, a server application program, using TCP, will typically use the following socket functions:

- **socket()**: This function creates a socket. The server creates a TCP socket with serverSocket=socket(AF_INET,SOCK_STREAM). The first parameter, AF_INET, indicates that the underlying network is using IPv4. The second parameter, SOCK_STREAM, indicates that the socket is a TCP socket.
- **bind()**: This function binds the socket to a specific port number so that clients can connect to the server using that port. The server associates the server port number, serverPort, with the socket with serverSocket.bind(('',serverPort)).
- **listen()**: This function puts the socket in listening mode, which means that it will wait for connection requests from clients. The server listens for TCP connection requests from the client with serverSocket.listen(1). This line has the server listen for TCP

connection requests from the client. The parameter specifies the maximum number of queued connections (at least 1).

- **accept()**: This function accepts a connection request from a client. When a client knocks on the welcoming door, the program invokes the accept() method for serverSocket, which creates a new socket in the server, called connectionSocket, dedicated to this particular client. The client and server then complete the handshaking, creating a TCP connection between the client's clientSocket and the server's connectionSocket.
- **send()**: This function sends data to the client. The send() function sends the string sentence through the server's socket and into the TCP connection.
- **recv()**: This function receives data from the client. With sentence = connectionSocket.recv(1024), the server extracts the TCP segment from the datagram.
- **close()**: This function closes the connection with the client. The TCP server can close the connection with the client using connectionSocket.close(). Note that since serverSocket remains open, another client can connect to the server after the connectionSocket is closed.

TCP Socket Functions: Client Side

On the client side, a client application program, using TCP, will typically use the following socket functions:

- **socket()**: This function creates a socket. Similar to the server side, the client creates a TCP socket with clientSocket = socket(AF_INET, SOCK_STREAM). The first parameter, AF_INET, indicates that the underlying network is using IPv4. The second parameter, SOCK_STREAM, indicates that the socket is a TCP socket.
- **connect()**: This function connects to a server. After creating the socket, the client initiates a three-way handshake and establishes a TCP connection with the server with the line clientSocket.connect((serverName,serverPort)). The parameter of the connect() method is the address of the server side of the connection.

- **send()**: This function sends data to the server. The client sends the string sentence through the socket and into the TCP connection using clientSocket.send(sentence).
- **recv()**: This function receives data from the server. The client uses the recv() function with modifiedSentence = clientSocket.recv(1024).
- **close()**: This function closes the connection with the server. The client closes the connection using clientSocket.close().

Note: This is not an exhaustive list of all socket functions for TCP servers and clients. There may be additional socket functions available.

Define Protocol. Explain key elements of a protocol.

Protocol Definition and Key Elements

A **protocol** is a set of rules that govern data communication between entities in different systems. These rules define what is communicated, how it is communicated, and when it is communicated. For two entities to communicate successfully, they must agree on a protocol. An analogy to this in human interaction is language - for two people to understand each other, they must speak the same language.

The key elements of a protocol are:

- **Syntax:** This refers to the structure or format of the data, meaning the order in which they are presented. The syntax of a protocol dictates how a message is organized so that the receiving device can interpret it correctly.
- **Semantics:** This refers to the meaning of each section of bits. It deals with how a particular pattern is interpreted and what action should be taken based on the interpretation. For example, does an address field identify the route to be taken or the final destination of the message?
- **Timing:** This refers to two characteristics: when data should be sent and how fast they can be sent. For instance, is transmission synchronous, with both sides operating at the same speed, or asynchronous, where the speed of transmission can vary?

The sources provide several examples of protocols, including TCP, UDP, HTTP, SMTP, FTP, DNS, ARP, RARP, ICMP, and IGMP.

From our previous conversation about the TCP/IP model, you know that different protocols are used at different layers of the model to accomplish different communication tasks.

How Attenuation,Distortion and Noise affects the information transmission.

Effects of Attenuation, Distortion, and Noise on Information Transmission

Attenuation, distortion, and noise are three causes of signal impairment that can affect the transmission of information over a transmission medium. Here is a detailed explanation of each:

Attenuation

Attenuation is the loss of a signal's energy due to the resistance of the medium. As a signal travels through a medium, some of its energy is converted to heat, effectively weakening the signal. This loss of signal strength can lead to:

- **Reduced signal quality:** A weakened signal may be more difficult for the receiver to interpret accurately, potentially resulting in data loss or corruption.
- **Limited transmission distance:** As attenuation increases with distance, signals can become too weak to be received reliably beyond a certain point.

To mitigate the effects of attenuation, **amplifiers** are used to boost the signal strength at intervals along the transmission path.

Decibels (dB) are used to measure the relative strengths of two signals, indicating either attenuation (loss of power, negative dB) or amplification (gain of power, positive dB). The formula for calculating decibel (dB) is:

$dB = 10 * \log10 (P2/P1)$

Where P1 and P2 are the powers of a signal at points 1 and 2, respectively.

Distortion

Distortion is the alteration of a signal due to the differing propagation speeds of each frequency component that makes up the signal. This difference in delay can cause the signal components to arrive at the receiver out of phase with each other, altering the shape of the composite signal. This distortion can result in:

- **Inter-symbol interference:** When different frequency components of a signal arrive at different times, they can interfere with each other, making it difficult for the receiver to accurately decode the signal.
- **Reduced data rate:** To combat distortion, the data rate may need to be reduced to ensure reliable signal reception.

Distortion is particularly problematic in **broadband transmission** because signals with a wide range of frequencies are transmitted.

Noise

Noise refers to external energy that corrupts a signal. This unwanted energy can be introduced by various sources, such as:

- **Thermal noise:** The random motion of electrons in a wire.
- **Induced noise:** Electromagnetic radiation from sources like motors and appliances.
- **Crosstalk:** Interference from adjacent wires.
- **Impulse noise:** High-energy spikes from sources like lightning or power lines.

Noise can cause:

- **Bit errors:** Noise can alter the value of bits, changing 0s to 1s and vice versa.
- **Reduced signal clarity:** Noise can make it difficult to distinguish the signal from the background noise, potentially leading to data loss or errors.

The **signal-to-noise ratio (SNR)** measures the relative strength of the desired signal to the unwanted noise. A higher SNR indicates a cleaner

signal with less noise interference. SNR is often expressed in decibels (dB) using the following formula:

SNRdB = 10 * log10 (SNR)

Impact on Information Transmission

Attenuation, distortion, and noise all contribute to the degradation of a signal as it travels through a transmission medium. This degradation can lead to:

- **Reduced data rates:** To ensure reliable communication in the presence of impairment, the data rate may need to be lowered.
- **Increased error rates:** Signal impairment increases the likelihood of bit errors, requiring error detection and correction mechanisms to ensure data integrity.
- **Retransmissions:** Corrupted data packets may need to be retransmitted, reducing the overall efficiency of data transmission.
- **Limited network performance:** All of the above factors can contribute to reduced network throughput, increased latency, and lower overall network performance.

Draw and explain in brief the five components of a data communication system.

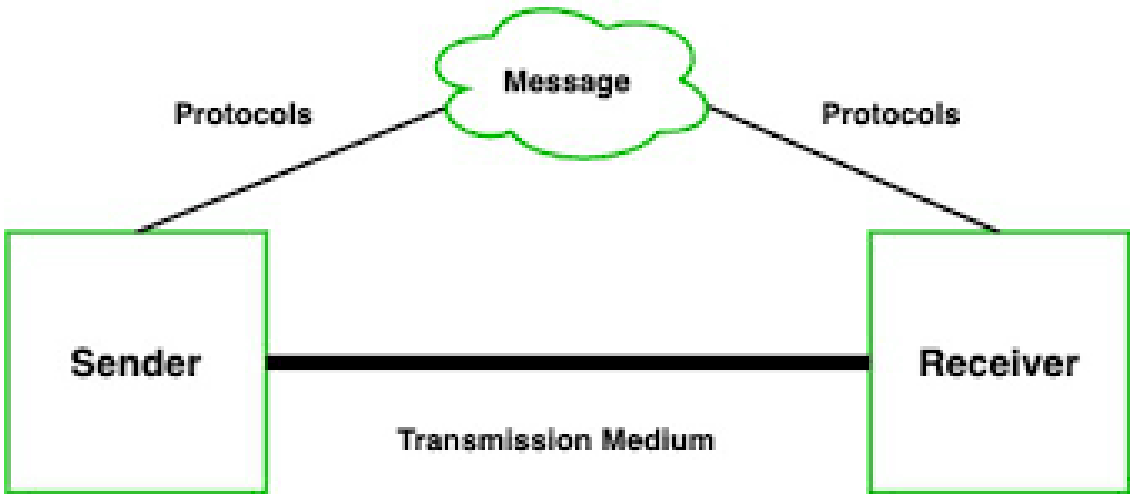
Components of a Data Communication System

The sources describe a data communication system as comprised of five components: **message, sender, receiver, transmission medium, and protocol**.

- **Message:** This is the information to be communicated. It can take many forms, such as text, numbers, pictures, audio, and video.
- **Sender:** This is the device that sends the data message. Examples include computers, workstations, telephones, and video cameras.
- **Receiver:** This is the device that receives the message. Examples include computers, workstations, telephones, and televisions.

- **Transmission Medium:** This is the physical path through which a message travels from sender to receiver. Examples include free space (wireless), metallic cable, and fiber optic cable. You also learned in our previous conversation that attenuation, distortion, and noise on the transmission medium can degrade signals and lead to reduced data rates, increased errors, and limited network performance.
- **Protocol:** This is a set of rules that govern data communications. As you learned in our conversation about protocols, they ensure that communicating devices are speaking the same "language" by defining the syntax, semantics, and timing of data exchange.

The sources emphasize that these five components are essential for data communication to occur. For example, even if two devices are physically connected by a transmission medium, they will not be able to communicate effectively without a shared protocol to govern the exchange of data. The sources use the analogy of two people trying to communicate in different languages to illustrate this point.



explain OSI Model.

Explanation of the OSI Model

The OSI (Open Systems Interconnection) model is a conceptual framework that standardizes the communication functions of a telecommunication or computing system without regard to its underlying internal structure and technology. In essence, it is a blueprint for how data is transmitted and received over a network. The OSI model is divided into seven distinct but interrelated layers, each responsible for a specific aspect of network communication. This layered approach offers several advantages, including:

- **Modularity:** Each layer can be developed and modified independently of the others, simplifying the design and implementation of network systems.
- **Flexibility:** Different protocols can be used at each layer to accommodate various network technologies and requirements.
- **Interoperability:** Systems from different vendors can communicate seamlessly as long as they adhere to the OSI standards.

The seven layers of the OSI model, from bottom to top, are:

1. **Physical Layer:** This layer is responsible for transmitting raw bits over a physical medium, such as copper wire, fiber optic cable, or radio waves. It defines the electrical and mechanical characteristics of the transmission medium, including:
    - Physical characteristics of interfaces and media: This includes specifying the type of connector, voltage levels, and signaling methods used at the interface between devices and the transmission medium.
    - Representation of bits: This defines how the digital data (0s and 1s) are encoded into signals, such as electrical or optical signals, for transmission over the physical medium.
    - Data rate: It sets the transmission speed, measured in bits per second (bps), which determines how fast data can be transmitted over the physical link.

- Synchronization of bits: This ensures that both the sending and receiving devices are synchronized at the bit level, meaning their internal clocks are aligned to correctly interpret the timing of the transmitted bits.
- Line configuration: It specifies whether the connection is point-to-point, where a dedicated link connects only two devices, or multipoint, where multiple devices share a single link.
- Physical topology: This defines the physical arrangement or layout of the network, such as mesh, star, bus, ring, or a hybrid combination of these topologies.
- Transmission mode: It defines the direction of data transmission between devices, which can be simplex (unidirectional), half-duplex (bidirectional but not simultaneous), or full-duplex (bidirectional and simultaneous).

2. **Data Link Layer:** This layer provides reliable transmission of data frames between two nodes connected by a physical link. It is responsible for:
   - Framing: Dividing the stream of bits from the network layer into manageable units called frames, adding header and trailer information to each frame to mark its beginning and end.
   - Physical addressing: Adding the physical addresses (MAC addresses) of the sender and receiver to the frame header to identify the specific devices on the link.
   - Flow control: Regulating the flow of data between the sender and receiver to prevent the sender from overwhelming the receiver with data faster than it can handle it.
   - Error control: Detecting and correcting errors that may occur during transmission over the physical link. This is typically achieved by adding error-detection codes to the frame, enabling the receiver to identify corrupted frames and request retransmission.
   - Media access control: In shared media topologies like Ethernet, the data link layer's MAC sublayer manages access

to the shared medium to prevent collisions and ensure orderly data transmission among multiple devices.

3. **Network Layer:** This layer is responsible for logical addressing and routing of data packets between different networks. Its key functions are:
    - Logical addressing: Assigning logical addresses (IP addresses) to devices on the network, allowing them to be uniquely identified across the internetwork, regardless of their physical location.
    - Routing: Determining the best path for data packets to travel from source to destination, potentially across multiple networks. This involves maintaining routing tables that store information about available routes and their associated costs.
4. **Transport Layer:** This layer provides reliable and efficient end-to-end data transfer between applications running on different hosts. It is responsible for:
    - Segmentation and reassembly: Dividing the data stream from the application layer into segments, adding sequence numbers, and ensuring they are delivered to the destination application in the correct order, even if they arrive out of order due to network conditions.
    - Flow control (end-to-end): Managing the data flow between applications on different hosts, ensuring the sending application does not overwhelm the receiving application.
    - Error control (end-to-end): Providing mechanisms for error detection and recovery across the entire end-to-end path between applications, ensuring data integrity even in the presence of unreliable network conditions.
    - Service-point addressing: Identifying the specific application or process on a host that data segments are intended for, enabling multiple applications on a single host to communicate concurrently over the network.

5. **Session Layer:** This layer manages the establishment, coordination, and termination of communication sessions between applications on different hosts. Its functions include:
    - Session establishment and termination: Initiating and closing communication sessions between applications, ensuring an orderly exchange of data.
    - Session management: Managing the dialog control, such as who transmits when and for how long, and synchronizing data exchange between applications.
    - Checkpointing and recovery: Providing mechanisms to insert checkpoints during a data transfer session, allowing recovery from errors or interruptions without having to resend the entire data stream.
6. **Presentation Layer:** This layer ensures that data exchanged between applications is in a format that both can understand. It provides services such as:
    - Data translation: Converting data between different formats used by different systems, ensuring compatibility and interoperability.
    - Data encryption and decryption: Providing security by encrypting data at the sender's side and decrypting it at the receiver's side, protecting sensitive information from unauthorized access.
    - Data compression and decompression: Compressing data to reduce the number of bits transmitted, saving bandwidth and speeding up data transfer, and decompressing it at the receiver's side.
7. **Application Layer:** This layer is the closest to the end user and provides network services to user applications. It provides services such as:
    - Network services access: Enabling applications to access network resources and services, such as file transfer, email, and web browsing.

- Application-specific protocols: Implementing protocols specific to various applications, such as HTTP for web browsing, SMTP for email, and FTP for file transfer.
- User interface: Providing a way for users to interact with the network and access network services.

Data Encapsulation and Decapsulation in the OSI Model

When data is transmitted over a network using the OSI model, it undergoes a process of encapsulation and decapsulation as it moves through the different layers.

- **Encapsulation:** At each layer of the sending device, a header (and sometimes a trailer) is added to the data unit received from the layer above. This process of adding control information to the data at each layer is called encapsulation. For example, at the transport layer, a header is added to the data received from the application layer to create a segment. This segment is then encapsulated in a packet at the network layer by adding a network layer header.
- **Decapsulation:** At the receiving device, the data unit moves up through the OSI layers. Each layer removes the headers and trailers added by its counterpart layer at the sending device. This process of removing the control information is called decapsulation. For instance, when the packet reaches the network layer of the receiving device, the network layer header is removed. The resulting segment is then passed up to the transport layer where the transport layer header is removed.

This process of encapsulation and decapsulation ensures that the data is delivered to the correct application and that each layer can process the data relevant to its functions.

While the OSI model is a powerful theoretical framework for understanding network communication, it's important to note that it's not a directly implemented protocol suite like TCP/IP. Nevertheless, the concepts and principles of the OSI model have significantly influenced the development of networking technologies and continue to be relevant in understanding modern network architectures.

What are the propagation time and transmission time for a 50 Mbyte message if the bandwidth of the network is 100 Mbps? Assume that the distance between the sender and receiver is 160,000 km and propagation speed of the medium 3 * 108 m/s. Calculate Latency in sec if the queueing delay is 0.10 sec and processing delay is 0.5 sec.

Calculating Propagation Time, Transmission Time, and Latency

You are asking to calculate the propagation time, transmission time, and latency for a 50 Mbyte message given certain network conditions. Here are the calculations:

1. Propagation Time

- **Definition:** Propagation time is the time it takes for a signal to travel from the sender to the receiver. It depends on the distance between the two and the propagation speed of the medium.
- **Formula:** Propagation time = Distance / Propagation speed
- **Calculation:**
    - Distance = 160,000 km = 160,000,000 meters
    - Propagation speed = $3 * 10^8$ m/s
    - Propagation time = 160,000,000 m / $3 * 10^8$ m/s = **0.533 seconds**

2. Transmission Time

- **Definition:** Transmission time is the time it takes for the entire message to be transmitted from the sender onto the transmission medium. It depends on the message size and the bandwidth of the channel.
- **Formula:** Transmission time = Message size / Bandwidth
- **Calculation:**
    - Message size = 50 MB = $50 * 8 * 10^6$ bits (1 byte = 8 bits)
    - Bandwidth = 100 Mbps = $100 * 10^6$ bps
    - Transmission time = ($50 * 8 * 10^6$ bits) / ($100 * 10^6$ bps) = **4 seconds**

3. Latency

- **Definition:** Latency, also known as total delay, is the total time it takes for a message to be sent and received. It includes propagation time, transmission time, and any other delays introduced by network devices like routers or switches.
- **Formula:** Latency = Propagation time + Transmission time + Queuing delay + Processing delay
- **Calculation:**
    - Propagation time = 0.533 seconds
    - Transmission time = 4 seconds
    - Queuing delay = 0.10 seconds
    - Processing delay = 0.5 seconds
    - Latency = 0.533 + 4 + 0.10 + 0.5 = **5.133 seconds**

In summary:

- **Propagation time:** 0.533 seconds
- **Transmission time:** 4 seconds
- **Latency:** 5.133 seconds

This example highlights the different components that contribute to the delay in delivering a message over a network. It's important to consider all these factors when designing and evaluating network performance, particularly for applications that are sensitive to delay, such as real-time communication or streaming.

Define following term w.r.t Line coding

a. Signal Element

b. Data Element

c. Data Rate

d. Signal Rate

e. Baseline Wandering

Definitions Related to Line Coding

- **Signal Element:** In digital data communications, signal elements carry data elements. A signal element is the **shortest unit of a digital signal** (measured in time).
- **Data Element:** Data elements are what we need to send, and signal elements are how we can send them. A data element is the **smallest entity that can represent information**. In digital data communications, the data element is the bit.
- **Data Rate:** The data rate, sometimes called the bit rate, is the **number of data elements (bits) sent in 1 s**. The unit is bits per second (bps).
- **Signal Rate:** The signal rate, sometimes called the pulse rate, modulation rate, or baud rate, is the **number of signal elements sent in 1 s**. The unit is the baud.
- **Baseline Wandering:** When decoding a digital signal, the receiver calculates a running average of the received signal power, called the baseline. The incoming signal power is evaluated against this baseline to determine the value of the data element. **A long string of 0s or 1s can make it difficult for the receiver to decode correctly by causing a drift in the baseline.** A good line coding scheme prevents baseline wandering.

Relationship Between Data Elements and Signal Elements

The relationship between data elements and signal elements is represented by the variable *r*. This variable represents the ratio of data elements to signal elements:

r = Data elements / Signal elements

For example, if *r* = 2, one signal element carries two data elements.

The sources provide a helpful analogy for understanding the relationship between data elements and signal elements:

Suppose each data element is a person who needs to be carried from one place to another. We can think of a signal element as a vehicle that can carry people. When $r = 1$, it means each person is driving a vehicle. When $r > 1$, it means more than one person is traveling in a vehicle (a carpool, for example). We can also have the case where one person is driving a car and a trailer ($r = 4/3$).

Compare between Router,Switch and Hub.

Comparing Routers, Switches, and Hubs

The sources provide information about routers and switches, primarily focusing on their use in Ethernet LANs, but they do not offer a direct comparison with hubs.

Here's what the sources say about routers and switches:

- **Routers:** Routers are network layer devices (layer 3) that forward packets based on logical addresses, such as IP addresses. They are crucial for connecting different networks, like LANs and WANs, and use routing tables to determine the best path for data transmission. These routing tables are often dynamic, updated by routing protocols. Routers offer advantages like efficient routing, traffic isolation, and security features like firewall protection, but they require IP configuration and can have higher processing times per packet compared to switches.
- **Switches:** Switches operate at the data link layer (layer 2) and forward frames based on MAC addresses. They learn MAC addresses automatically and build their switching tables dynamically. Compared to hubs, switches offer improved performance by eliminating collisions and allowing for heterogeneous links with different speeds and media. While switches are plug-and-play and generally faster than routers in processing, they lack the sophisticated routing capabilities of routers and are limited to spanning tree topologies.

Hubs are not directly discussed in the provided sources. However, information from outside the sources indicates that hubs, unlike routers and switches, do not perform any intelligent forwarding. They simply

broadcast any data received on one port to all other ports. This can lead to collisions and reduced network efficiency, especially as more devices are added.

Based on information from outside the sources, here's a table comparing routers, switches, and hubs:

Feature	Router	Switch	Hub
Layer	Network Layer (Layer 3)	Data Link Layer (Layer 2)	Physical Layer (Layer 1)
Forwarding Basis	IP Address	MAC Address	None (broadcasts data to all ports)
Collision Domains	Separate collision domain for each port	Separate collision domain for each port	Single collision domain for all devices connected
Speed	Slower than switches due to higher layer processing	Faster than routers, as they only process up to Layer 2	Typically the slowest option
Cost	Generally the most expensive	Mid-range cost	Least expensive
Configuration	Requires IP configuration for connected devices	Plug-and-play, learns MAC addresses automatically	No configuration needed
Applications	Connecting different networks, providing routing and security features	Connecting devices within a LAN, segmenting a network to improve performance	Simple and small networks where cost is a major factor and performance is not critical

Please note that the information about hubs in the table above is not derived from the sources you provided. I recommend you independently verify this information from reliable sources.

How 5 components of data communication relate each other to transfer a data from source to destination.

Five Components of Data Communication and Their Interrelation

The five components of a data communication system are: **message, sender, receiver, transmission medium, and protocol**. These components work together to ensure the successful transfer of data from its source to its intended destination. Here's how they relate to each other:

- **Message:** The message is the **information (data) to be communicated**. This could be in the form of text, numbers, pictures, audio, or video. The message acts as the payload that the data communication system aims to deliver.
- **Sender:** The sender is the device that **sends the data message**. This could be a computer, workstation, telephone handset, video camera, etc.. The sender is responsible for preparing the message for transmission, which often involves breaking it down into smaller units.
- **Receiver:** The receiver is the device that **receives the message**. Examples include computers, workstations, telephones, and televisions. The receiver is responsible for accepting the transmitted message and reconstructing it into a usable form.
- **Transmission Medium:** The transmission medium is the **physical path over which the message travels from sender to receiver**. Examples include free space, metallic cable, and fiber optic cable. The choice of medium depends on factors like distance, desired data rate, and cost.
- **Protocol:** The protocol is a **set of rules that govern data communication**, representing an agreement between communicating devices. These rules are essential to ensure orderly and understandable communication, much like a common language is needed for people to communicate effectively. Protocols cover aspects like:
    - **Message Formatting:** Defining how the message is organized, including headers, data, and trailers.

- **Error Control:** Mechanisms to detect and potentially correct errors that may occur during transmission.
- **Flow Control:** Regulating the rate of data transmission to prevent the receiver from being overwhelmed.
- **Addressing:** Specifying how devices are identified and located on the network.

How the Components Work Together:

1. The **sender** wants to transmit a **message** to the **receiver**.
2. The sender encodes the message into a suitable format for transmission over the chosen **transmission medium**.
3. The message, packaged according to the **protocol**, is sent over the transmission medium.
4. The **receiver**, following the same **protocol**, receives the transmitted signal.
5. The receiver decodes the signal and extracts the original **message**.

Example:

Consider sending an email using a smartphone.

- **Message:** The email content you compose.
- **Sender:** Your smartphone.
- **Receiver:** The recipient's email server.
- **Transmission Medium:** Wireless network (e.g., Wi-Fi) and the internet (a network of networks).
- **Protocol:** A combination of protocols, including TCP/IP for internet communication and SMTP for email transmission.

Importance of Interrelation:

The successful transfer of data relies on the seamless interaction of all five components. If any component fails or does not adhere to the established protocol, the communication will likely break down. For example, if the sender and receiver are using different protocols, they won't understand each other, much like two people speaking different languages. Similarly, a

noisy transmission medium can corrupt the message, making it unintelligible to the receiver.

Explain distributed processing and explain network criterion for distributed processing.

Distributed Processing and Network Criteria

The sources provide a definition of distributed processing and list important criteria for evaluating networks, particularly in the context of distributed processing.

Distributed Processing

- **Definition:** Distributed processing is a network paradigm where a task is divided among multiple computers instead of relying on a single, centralized machine.
    - Instead of one powerful computer handling the entire process, separate, interconnected computers, often personal computers or workstations, each handle a portion of the task.
- **Example:** Consider sending an email. With distributed processing, various computers and servers work together: your computer composes the message, your email provider's server handles sending the email, and the recipient's email server receives and stores it. No single machine is responsible for the entire process.

Network Criteria for Distributed Processing

The sources highlight three essential criteria for evaluating the effectiveness of networks, especially those employing distributed processing: **performance, reliability, and security.**

1. Performance

- **Definition:** Network performance refers to how well data is transferred within a network. This can be measured by:

    - **Transit Time:** The time a message takes to travel from one device to another.
    - **Response Time:** The time between an inquiry being sent and a response being received.
- **Factors Affecting Performance:** Several factors influence network performance, including:
    - **Number of Users:** More users sharing the network's resources can lead to slower speeds.
    - **Transmission Medium:** The physical path data travels through (e.g., copper cable, fiber optic cable, wireless) affects speed and reliability.
    - **Hardware Capabilities:** The processing power and memory of connected devices play a role.
    - **Software Efficiency:** Well-designed network software can optimize data transfer and improve performance.
- **Key Metrics:** Two primary metrics are used to evaluate network performance:
    - **Throughput:** The amount of data that can be transmitted per unit of time (e.g., bits per second). Higher throughput generally means faster data transfer rates.
    - **Delay:** The time it takes for data to travel from the source to the destination. Lower delay is crucial for applications sensitive to latency, such as video conferencing or online gaming.
- **Tradeoffs:** There's often a tradeoff between throughput and delay. For instance, sending more data might increase throughput, but it could also increase congestion, leading to higher delays.

2. Reliability

- **Definition:** Reliability relates to the network's ability to consistently and accurately deliver data.
- **Factors Affecting Reliability:**

    - **Frequency of Failure:** How often network components (e.g., links, routers) experience problems.
    - **Recovery Time:** The time it takes for a network link to restore functionality after a failure.
    - **Robustness:** The network's ability to withstand and recover from significant disruptions or catastrophic events.
- **Importance in Distributed Processing:** A reliable network is crucial for distributed processing because any interruption or data loss can disrupt the entire task execution, potentially leading to inconsistencies or failures.

3. Security

- **Definition:** Security involves protecting the network and its data from unauthorized access, damage, and disruption.
- **Key Aspects of Network Security:**
    - **Data Confidentiality:** Preventing unauthorized parties from reading or accessing sensitive information.
    - **Data Integrity:** Ensuring that data is not tampered with during transmission or storage.
    - **Access Control:** Restricting network access to authorized users and devices.
    - **Data Protection and Recovery:** Implementing measures to safeguard data against loss and having procedures to recover data in case of breaches or failures.
- **Importance in Distributed Processing:** Since distributed processing often involves handling sensitive information across multiple devices, robust security measures are paramount to protect data integrity, ensure privacy, and prevent unauthorized access or malicious activities.

The sources emphasize that these criteria are fundamental for assessing the effectiveness and efficiency of networks, especially those used for distributed processing where tasks are distributed across multiple, interconnected computers.

For N device in a network what is the number of cable links required for

a. Bus

b. star

c. Ring Mesh.

Cable Links Required for N Devices in Different Topologies

The sources describe various network topologies, including **bus, star, ring, and mesh**, and explain how they differ in terms of their physical or logical arrangement. Based on these descriptions, here's a breakdown of the number of cable links required for N devices in each topology:

a. Bus Topology

- **Cable Links Required: 1**
- **Explanation:** A bus topology uses a single cable, often referred to as the backbone or segment, to which all devices are connected. This single cable acts as the shared communication medium.

b. Star Topology

- **Cable Links Required: N**
- **Explanation:** In a star topology, each device has a dedicated point-to-point link connecting it to a central hub or switch. Since there are N devices, you need N individual cable links to connect them all to the central point.

c. Ring Topology

- **Cable Links Required: N**
- **Explanation:** A ring topology arranges devices in a closed loop or circular pattern. Each device has two connections: one to its immediate neighbor on each side. With N devices, you have N connections, forming the ring structure.

d. Mesh Topology

- **Cable Links Required: N(N-1)/2**

- **Explanation:** In a full mesh topology, every device has a dedicated point-to-point link to every other device in the network. To calculate the number of links, consider that each of the N devices needs N-1 connections to connect to the other devices. However, this counts each link twice (once for each device it connects). Therefore, we divide by 2, resulting in the formula: N(N-1)/2.

What is the bandwidth of signal that can be decompose in to five sine waves with frequency 10,20,50,100,200 Hz? And all Peak amplitude are same.

Bandwidth Calculation

The bandwidth of a composite signal is **the difference between the highest and lowest frequencies** present in that signal.

- In your query, the highest frequency component is 200 Hz.
- The lowest frequency component is 10 Hz.

Therefore, the bandwidth of the signal is:

Bandwidth = 200 Hz - 10 Hz = 190 Hz

Even though all the sine waves have the same peak amplitude, it doesn't affect the bandwidth calculation. Bandwidth is only concerned with the **range of frequencies** present in the signal.

What are the types of sockets? Explain various socket primitives used in connection oriented client server approach.

Types of Sockets and Socket Primitives

The sources provide information on the types of sockets and the primitives used in connection-oriented client-server communication:

1. Types of Sockets

While the sources don't explicitly categorize sockets into distinct "types," they highlight a key distinction based on the underlying transport layer protocol:

- **TCP Sockets (SOCK_STREAM):** These sockets use the Transmission Control Protocol (TCP).
    - TCP provides a **connection-oriented**, **reliable** byte stream service. This means:
        - A connection must be established between the client and server before data exchange.
        - Data is delivered in order, without errors, and with flow control to prevent overwhelming the receiver.
    - TCP sockets are identified by a **four-tuple:** source IP address, source port number, destination IP address, and destination port number. This enables multiplexing and demultiplexing, allowing multiple applications on a single host to use TCP simultaneously.
- **UDP Sockets (SOCK_DGRAM):** These sockets utilize the User Datagram Protocol (UDP).
    - UDP provides a **connectionless**, **unreliable** datagram service.
        - No connection setup is required; data is sent as independent packets (datagrams).
        - There's no guarantee of delivery, order, or protection against data corruption.
    - UDP sockets are identified by a **two-tuple:** destination IP address and destination port number.

2. Socket Primitives in Connection-Oriented (TCP) Client-Server Approach

Socket primitives are functions or methods provided by the operating system that allow applications to create, manage, and use sockets for network communication. Here are key socket primitives used in the context of a connection-oriented client-server model, primarily using TCP sockets:

Server-Side Primitives:

- socket(AF_INET, SOCK_STREAM): Creates a TCP socket on the server, specifying the address family (IPv4 in this case) and socket type (SOCK_STREAM for TCP).
- bind((' ', serverPort)): Associates the server socket with a specific port number on the server's IP address. The '' indicates that the socket will accept connections on any available local IP address.
- listen(1): Puts the server socket in a listening state, ready to accept incoming connection requests from clients. The parameter (1 in this case) specifies the maximum number of queued connection requests.
- accept(): Blocks (waits) until a client connection request arrives. When a request is received, accept() creates a new socket, called the connection socket, specifically for communication with that client.
- recv(1024): Reads data arriving on the connection socket. The parameter (1024 in this case) specifies the maximum amount of data to read at once.
- send(data): Sends data over the connection socket to the connected client.
- close(): Closes the connection socket, terminating the TCP connection with the client.

Client-Side Primitives:

- socket(AF_INET, SOCK_STREAM): Similar to the server, the client creates a TCP socket.
- connect((serverName, serverPort)): Initiates a TCP connection to the server's listening socket, specified by the server's hostname or IP address (serverName) and port number (serverPort). This triggers the three-way handshake process.
- send(data): Sends data to the server through the established TCP connection.
- recv(1024): Reads data received from the server.

- close(): Closes the client socket, terminating the connection with the server.

Key Points about Socket Primitives in Connection-Oriented Communication:

- **Connection Establishment (Three-way Handshake):** TCP requires a three-way handshake to establish a connection, ensuring both sides are ready for data exchange.
- **Ordered and Reliable Data Transfer:** TCP guarantees that data will be delivered in the order sent and without errors.
- **Full-Duplex Communication:** Once the TCP connection is established, the client and server can send and receive data simultaneously (full-duplex) over the same connection.

A computer monitor has a resolution of 1600 by 1200 pixels. Each pixel uses 65,536 colors. If 30 frames are displayed per second on the screen then how many bits are needed to send the complete contents of a screen of 5 second?

Calculating Bits for Screen Display

To calculate the total number of bits needed, you can follow these steps:

1. **Bits per pixel:** Since each pixel can display 65,536 colors, you need to find how many bits can represent that many unique values. The formula for this is 2^bits = number of colors. In this case:
    - 2^bits = 65,536
    - bits = log2(65,536) = 16 bits per pixel
2. **Bits per frame:** With a resolution of 1600x1200 pixels and 16 bits per pixel, the total bits per frame is:
    - 1600 pixels * 1200 pixels * 16 bits/pixel = 30,720,000 bits per frame
3. **Bits per second:** At a frame rate of 30 frames per second, the bit rate is:

    - 30,720,000 bits/frame * 30 frames/second = 921,600,000 bits per second (bps)
4. **Total bits for 5 seconds:** Finally, the total number of bits for 5 seconds of video is:
    - 921,600,000 bps * 5 seconds = **4,608,000,000 bits**

Therefore, you would need **4,608,000,000 bits** to send the complete contents of the screen for 5 seconds. This calculation assumes no compression is used. Video data is often compressed to reduce the number of bits required for transmission or storage.

Explain the different modes of communication with neat and clean diagram. Specify the real time applications of the same.

Modes of Communication

The sources describe different ways data can flow between devices, known as **transmission modes** or **data flow**. Here's an explanation of these modes, along with diagrams and real-time applications:

1. Simplex Mode

- **Explanation:** In simplex mode, communication is unidirectional, meaning data flows in only one direction. One device acts solely as a transmitter (sender), while the other acts solely as a receiver. This is analogous to a one-way street.
- **Diagram:**
- [Sender] -----> Data Flow -----> [Receiver]
- **Real-Time Applications:**
    - **Keyboards and traditional monitors:** Keyboards only send input, and monitors only receive output.
    - **Radio and television broadcasting:** Signals are transmitted from a central station to receivers.
    - **Security cameras:** Video data is sent from the camera to a monitoring station.

2. Half-Duplex Mode

- **Explanation:** Half-duplex mode allows for bidirectional communication, but data can only flow in one direction at a time. This is similar to a one-lane road with traffic allowed in both directions; cars traveling in opposite directions must take turns. Each device can both transmit and receive, but not simultaneously.
- **Diagram:**
- [Device 1] <----- Data Flow at Time 1 -----> [Device 2]
- [Device 1] -----> Data Flow at Time 2 -----> [Device 2]
- **Real-Time Applications:**
    - **Walkie-talkies:** Only one person can speak at a time, and the other must listen.
    - **Hub-based computer networks:** While hubs allow multiple devices to connect, they operate in half-duplex mode. Only one device can send data at a time, leading to collisions if multiple devices try to transmit simultaneously.

3. Full-Duplex Mode

- **Explanation:** Full-duplex mode enables simultaneous two-way communication, with data flowing in both directions concurrently. This is like a two-lane road where traffic can flow in both directions simultaneously. Both devices can transmit and receive at the same time.
- **Diagram:**
- [Device 1] <------> Data Flow <------> [Device 2]
- **Real-Time Applications:**
    - **Telephone networks:** Both parties can talk and listen simultaneously.
    - **Modern computer networks (using switches):** Switches enable full-duplex communication, allowing devices to send and receive data concurrently without collisions.

    - **Video conferencing:** Participants can send and receive audio and video data simultaneously.

Key Points about Data Flow and Its Significance:

- The choice of transmission mode depends on the application's requirements and the capabilities of the communication medium.
- Simplex mode is the simplest but most limited, suitable for applications where data only needs to flow in one direction.
- Half-duplex mode is more versatile but can be inefficient if frequent back-and-forth communication is required.
- Full-duplex mode provides the highest capacity and efficiency for two-way communication but often requires more complex hardware or protocols.
- The evolution of computer networks from hubs to switches reflects the shift towards full-duplex communication for increased performance.

What is the theoretical capacity of a channel in each of the following cases? a. Bandwidth: 45 KHz b. Bandwidth: 250 KHz c. Bandwidth: 10 MHz SNRdB = 50 SNRdB = 8 SNRdB = 30

Theoretical Channel Capacity Calculations

The sources explain how to calculate the theoretical capacity of a channel using the Shannon Capacity formula, which takes into account both bandwidth and signal-to-noise ratio (SNR). Here's the breakdown of the calculation for each case:

Formula:

Capacity (C) = Bandwidth (B) * log2(1 + SNR)

Where:

- **Capacity (C):** The maximum theoretical data rate in bits per second (bps).

- **Bandwidth (B):** The bandwidth of the channel in Hertz (Hz).
- **SNR:** The signal-to-noise ratio, expressed as a linear power ratio (not in decibels).

Converting SNRdB to SNR:

The sources provide the SNR values in decibels (SNRdB). To use the formula, you need to convert SNRdB to a linear power ratio (SNR) using the following equation:

SNR = 10^(SNRdB / 10)

Calculations:

a. Bandwidth: 45 KHz, SNRdB = 50

1. **Convert SNRdB to SNR:**
    - SNR = 10^(50 / 10) = 100,000
2. **Calculate Capacity (C):**
    - C = 45,000 Hz * log2(1 + 100,000)
    - C ≈ 45,000 Hz * 16.61 bps/Hz
    - **C ≈ 747,450 bps (approximately 747.45 kbps)**

b. Bandwidth: 250 KHz, SNRdB = 8

1. **Convert SNRdB to SNR:**
    - SNR = 10^(8 / 10) ≈ 6.31
2. **Calculate Capacity (C):**
    - C = 250,000 Hz * log2(1 + 6.31)
    - C ≈ 250,000 Hz * 2.66 bps/Hz
    - **C ≈ 665,000 bps (approximately 665 kbps)**

c. Bandwidth: 10 MHz, SNRdB = 30

1. **Convert SNRdB to SNR:**
    - SNR = 10^(30 / 10) = 1000
2. **Calculate Capacity (C):**

- C = 10,000,000 Hz * log2(1 + 1000)
- C ≈ 10,000,000 Hz * 9.97 bps/Hz
- **C ≈ 99,700,000 bps (approximately 99.7 Mbps)**

Key Takeaways from the Shannon Capacity Formula:

- **Bandwidth and SNR Limits:** The Shannon Capacity formula demonstrates that the theoretical data rate is limited by both the bandwidth of the channel and the signal-to-noise ratio.
- **Impact of Noise:** As noise increases (lower SNR), the channel capacity decreases, even if the bandwidth remains the same.
- **Logarithmic Relationship:** The capacity increases logarithmically with SNR. This means that large increases in SNR are required to achieve substantial gains in capacity.

i) Discuss the need of Digital signal to Digital signal conversion techniques.

ii) ii) Explain any four techniques with their advantages and disadvantages.

Digital Signal to Digital Signal Conversion Techniques

i) Need for Conversion

The sources provide insights into why digital-to-digital conversion techniques are essential in data communications:

- **Representing Digital Data as Signals:** Digital data, stored in computers as bits (0s and 1s), needs to be transformed into electrical or optical signals for transmission over physical media (). Digital-to-digital conversion techniques provide the means to represent these bits as distinct signal elements that can be transmitted and received ().
- **Addressing Signal Impairments:** Transmission media are not perfect (). Attenuation, distortion, and noise can degrade signals as

they travel (). Some digital-to-digital conversion techniques incorporate features to mitigate these impairments. For example:

    - **Synchronization:** Techniques like Manchester encoding ensure frequent signal transitions, helping the receiver's clock stay synchronized with the sender's clock, even in the presence of long strings of 0s or 1s ().
    - **DC Component:** Some schemes avoid a DC component (a non-zero average voltage level), which can cause issues with certain transmission media ().

- **Bandwidth Considerations:** Digital signals, especially those with sharp transitions, inherently have a wide bandwidth (). Digital-to-digital conversion techniques can shape the signal's spectrum to:
    - **Match Channel Characteristics:** Allow transmission over band-limited channels by concentrating signal energy in specific frequency ranges ().
    - **Improve Bandwidth Efficiency:** Some techniques aim to represent data using fewer signal transitions, effectively reducing the required bandwidth ().

ii) Four Digital-to-Digital Conversion Techniques

Here's an explanation of four digital-to-digital conversion techniques, drawing on the sources:

1. **Non-Return-to-Zero Level (NRZ-L)**
    - **Explanation:** NRZ-L is a simple encoding scheme where a high voltage level represents a binary 1, and a low voltage level represents a binary 0 (). The signal level remains constant during the bit interval.
    - **Advantages:**
        - **Easy Implementation:** NRZ-L is straightforward to implement electronically.
        - **Good Bandwidth Efficiency:** It has a relatively low bandwidth requirement compared to some other techniques.

- **Disadvantages:**
    - **Synchronization Issues:** Long sequences of 0s or 1s can cause the receiver's clock to lose synchronization with the sender ().
    - **DC Component:** The presence of a DC component can be problematic for some transmission media.

2. **Manchester Encoding**
    - **Explanation:** Manchester encoding divides each bit interval into two halves. A transition occurs at the middle of the bit interval. A high-to-low transition represents a binary 0, while a low-to-high transition represents a binary 1 ().
    - **Advantages:**
        - **Self-Clocking:** The guaranteed transition at the middle of each bit provides inherent clock synchronization, eliminating the issue of synchronization errors caused by long strings of 0s or 1s ().
    - **Disadvantages:**
        - **Higher Bandwidth:** The requirement for a mid-bit transition doubles the signal rate compared to NRZ-L, leading to a higher bandwidth requirement ().
3. **Differential Manchester Encoding**
    - **Explanation:** Differential Manchester encoding combines aspects of NRZ-I (Non-Return-to-Zero Inverted) and RZ (Return-to-Zero). There's always a transition at the middle of the bit interval for synchronization. However, the presence or absence of a transition at the beginning of the bit interval determines the bit value. A transition at the beginning signifies a binary 0, while no transition indicates a binary 1 ().
    - **Advantages:**
        - **Self-Clocking:** Similar to Manchester encoding, the mid-bit transition ensures synchronization ().

        - **Error Detection:** The transition at the beginning of the bit interval provides some level of error detection. If a single transition is lost, the receiver can often infer the correct bit values from the remaining transitions.
    - **Disadvantages:**
        - **Higher Bandwidth:** Like Manchester, it requires a higher bandwidth than NRZ schemes due to the additional transitions ().

4. **2B1Q (Two Binary, One Quaternary)**
    - **Explanation:** 2B1Q is a multilevel encoding scheme where groups of 2 bits are mapped to one of four voltage levels (). Each signal element represents 2 bits, increasing data density.
    - **Advantages:**
        - **Improved Bandwidth Efficiency:** By encoding 2 bits per signal element, 2B1Q reduces the required bandwidth compared to techniques that transmit one bit per signal element.
        - **Used in DSL:** It's employed in DSL technologies to provide high-speed internet access over telephone lines ().
    - **Disadvantages:**
        - **Complexity:** The encoding and decoding process is more complex than simpler schemes.
        - **Susceptibility to Noise:** Multilevel schemes like 2B1Q are generally more susceptible to noise than two-level techniques because the voltage differences between levels are smaller, making it harder to distinguish between them in the presence of noise.