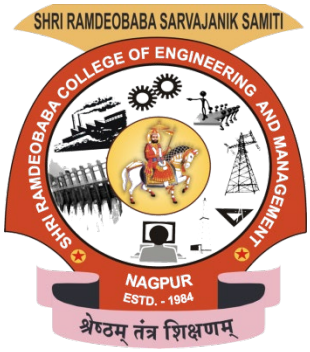
Computer Networks Lab (MCP547)

M.C.A.-Semester-II



Session: 2023-24

Shri Ramdeobaba College of Engineering and Management, Gittikhadan,Katol Road, Nagpur 440013 (M.S.)

Shri Ramdeobaba College of Engineering and Management, Nagpur
Department of Computer Application
SYLLABUS OF SEMESTER-II, M.C.A. (MASTER IN COMPUTER APPLICATION)

Index

Sr. No.	Aim/Practical Title	Date of Submission	Remark
1	To study network devices and communication channels: A) Network Devices: Switches, Hubs, Routers, Repeaters, NIC. B) Communication Channels: Wired medium - CAT5/CAT6 cables, Coaxial cable, Fiber optic + Connectors. Wireless medium - Radio, Microwave, Infrared. Frequency band. C) IP address (Classification of IP address, Sub netting, Super netting)	26-01-2024	
2	Implement the Client-Server communication. i. TCP communication. ii. UDP communication	16-02-2024	
3	Implementation of various framing techniques using client server communication i. Byte Stuffing ii. Bit Stuffing	29-02-2024	
4	Implementation of Error detection algorithm i. CRC (Cyclic Redundancy Check) Method. ii. Hamming distance (code) method for detection/correction.	15-03-2024	

Name of Student: Jayesh Lalit Nandanwar
Division: 2
Roll Number: 26
Batch No: 2

Practical 1

Name of Program: To study network devices and communication channels:

A) Network Devices:
Switches, Hubs, Routers, Repeaters, NIC.

B) Communication Channels:
Wired medium - CAT5/CAT6 cables, Coaxial cable, Fiber optic + Connectors.
Wireless medium - Radio, Microwave, Infrared. Frequency band.

C) IP address (Classification of IP address, Sub netting, Super netting)

Source Code/Theory:

A) Network Devices

1) Switches:

a) Characteristics:

- Switches are hardware devices used to connect multiple devices within a local area network (LAN).
- Operating at the data link layer (Layer 2) of the OSI model, switches use MAC addresses for frame forwarding.
- Switches can be categorized as managed (configurable) or unmanaged (plug-and-play).

b) Diagram:

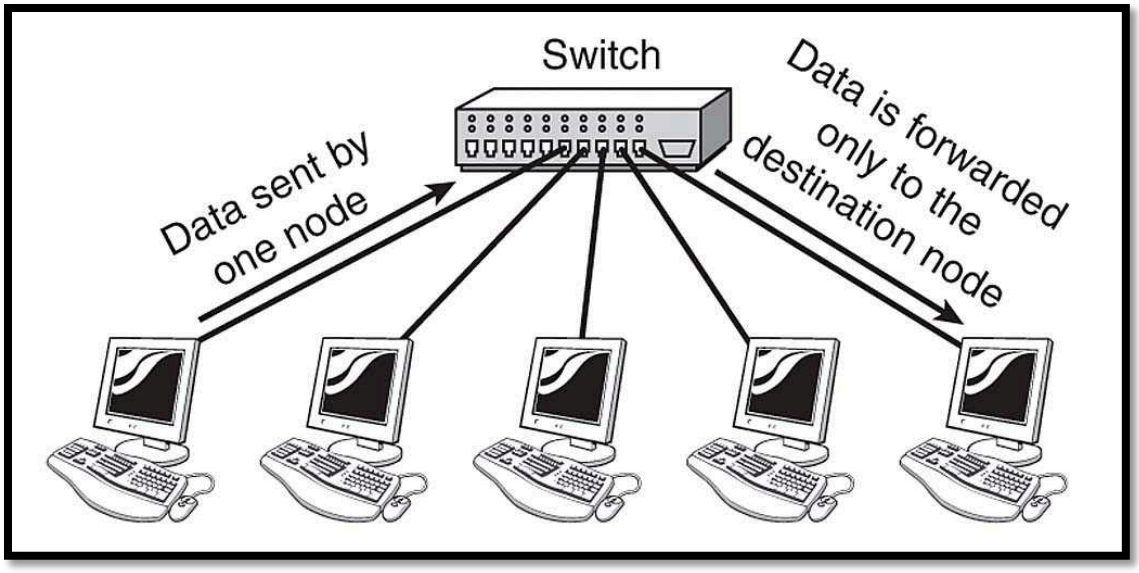


Fig: Switches in Networking.

c) Advantages:
- Dedicated bandwidth to each connected device enhances network performance.
- Network segmentation improves security and reduces congestion.
- Managed switches offer features like VLANs and Quality of Service (QoS).
- Transmission mode is full duplex, i.e. communication in the channel occurs in both the directions at the same time. Due to this, collisions do not occur.
- Switches can perform some error checking before forwarding data to the destined port.
- The number of ports is higher – 24/48

d) Disadvantages:
- Switches may be more expensive than simpler network devices like hubs.
- Managed switches require additional configuration and maintenance.

e) Limitations:
- Limited number of ports can be a constraint for larger networks.

f) Technical Details:
- Utilizes packet switching for data forwarding.
- Comes in various speeds, including 10/100/1000 Mbps or higher (10/25/40/100 Gbps).

2) Hubs:

a) Characteristics:
- A hub is a common connection point, also known as a network hub, which is used for connection of devices in a network.
- It works as a central connection for all the devices that are connected through a hub.
- Hubs are basic networking devices operating at the physical layer (Layer 1) of the OSI model.
- They broadcast data to all connected devices without intelligence about the destination.
- Typically, hubs are unmanaged devices.

b) Diagram:

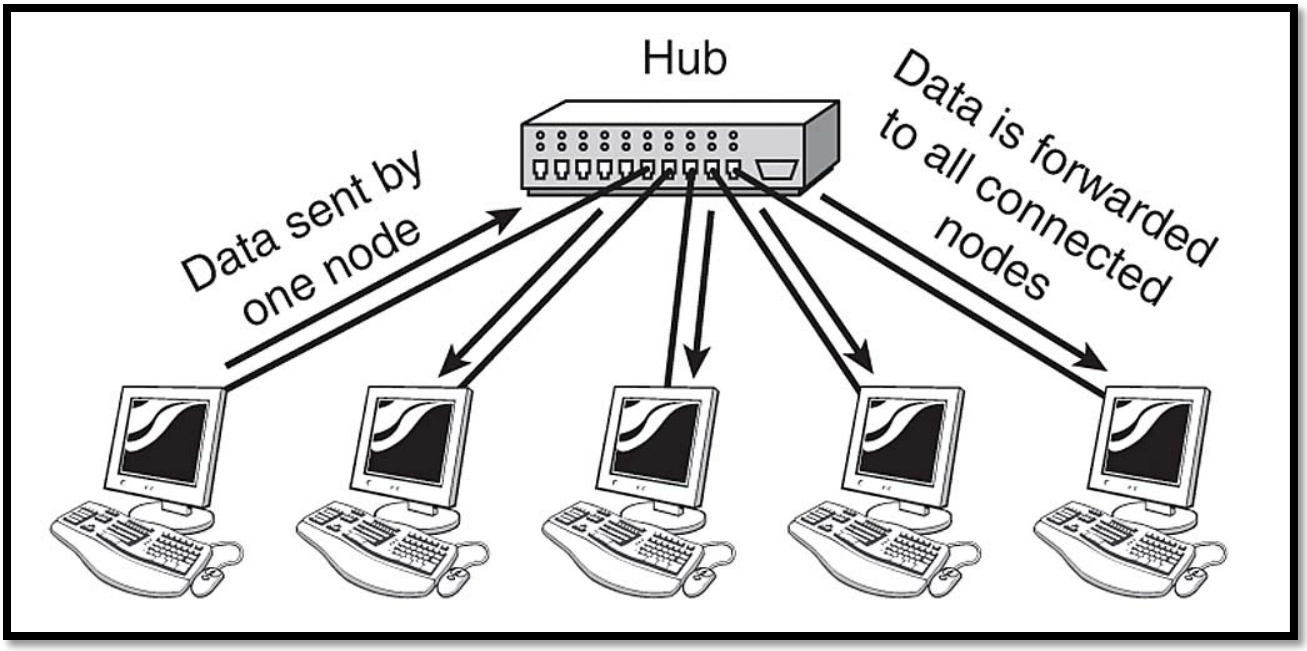


Fig: Hubs in Networking.

c) Advantages:
- Hubs are cost-effective.
- The use of a hub does not impact on the network performance.
- Easy to install and require minimal configuration.

d) Disadvantages:
- Network congestion and decreased performance due to broadcast nature.
- Lack security features and network segmentation.

e) Limitations:
- Limited number of ports.

f) Technical Details:
- Uses broadcast switching for data forwarding.
- Operates at a fixed speed, e.g., 10 Mbps or 100 Mbps.

3) Routers:

a) Characteristics:
- Routers connect multiple networks, operating at the network layer (Layer 3) of the OSI model.
- The router is a physical or virtual internetworking device that is designed to receive, analyze, and forward data packets between computer networks.
- A router examines a destination IP address of a given data packet, and it uses the headers and forwarding tables to decide the best way to transfer the packets.
- They use IP addresses for data routing.
- Can be managed or unmanaged.
- A router is used in LAN (Local Area Network) and WAN (Wide Area Network) environments. It shares information with other routers in networking.

b) Diagram:

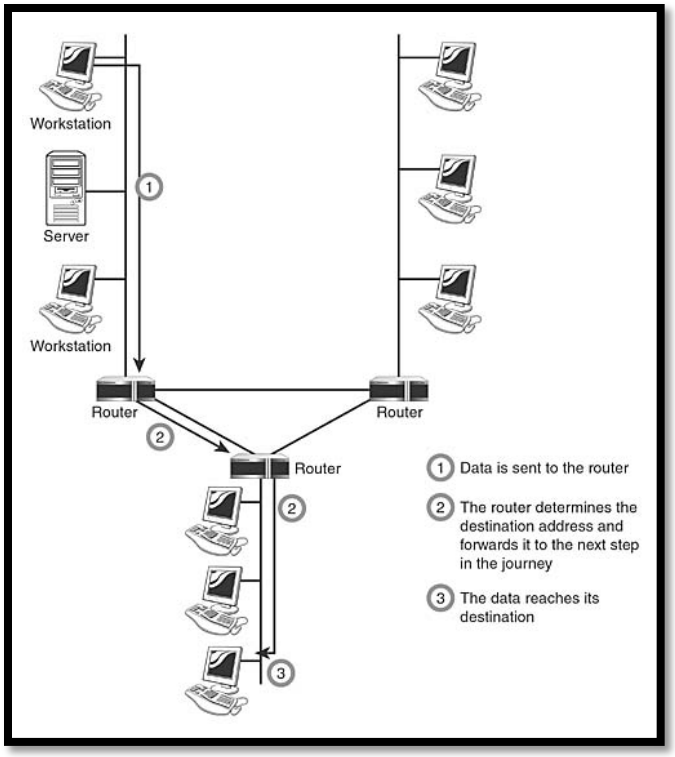


Fig: Routers in Networking.

c) Advantages:
- Provide security and segmentation between networks.
- Prioritize traffic using Quality of Service (QoS).
- Can connect different types of networks, such as wired and wireless.

d) Disadvantages:
- Routers may be more expensive and require more configuration.
- Managed routers need ongoing maintenance.

e) Limitations:
- Limited number of ports.

f) Technical Details:
- Uses packet switching for data forwarding.
- Available at various speeds like 10/100/1000 Mbps or higher (10/25/40/100 Gbps).

4) Repeaters:

a) Characteristics:
- Used to extend the range of a network, operating at the physical layer.
- Repeaters amplifies the attenuated signal and then retransmits it. Digital repeaters can even reconstruct signals distorted by transmission loss. So, repeaters are popularly incorporated to connect between two LANs thus forming a large single.
- Regenerates and amplifies signals to overcome signal attenuation.
- Typically unmanaged devices.

b) Diagram:

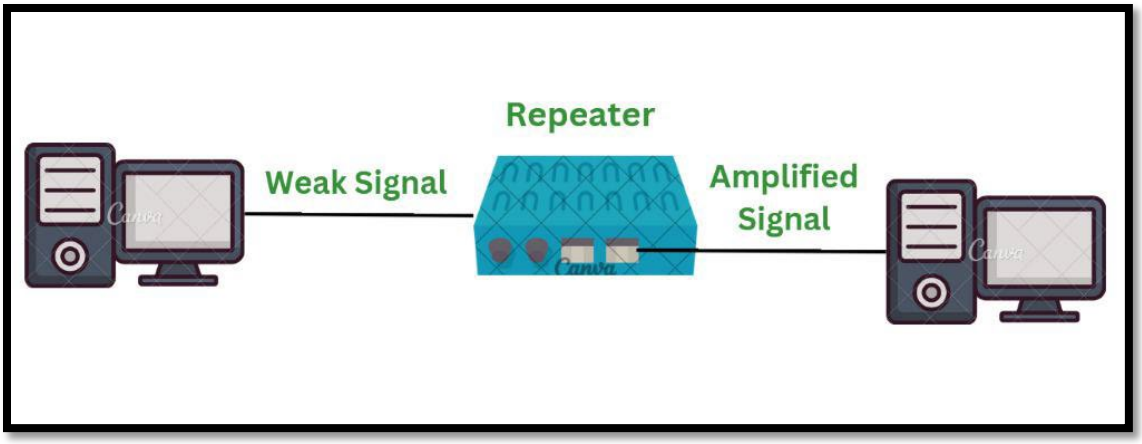


Fig: Repeaters in Networking.

c) Advantages:
- Extends network range without additional cabling.
- Easy installation and use.
- Repeaters don't require any processing overhead. The only time they need to be investigated is in case of degradation of performance.
- They can connect signals using different types of cables.

d) Disadvantages:
- Can cause congestion and slow down network performance.
- Lack security and segmentation features.
- Most networks have limitations upon the number of repeaters that can be deployed.

e) Limitations:
- Limited range suitable only for short distances.

f) Technical Details:
- Regenerates and amplifies signals at the physical layer.

5) NIC (Network Interface Card):

a) Characteristics:
- NIC is a hardware component connecting a device to a network, operating at the data link layer.
- Uses MAC addresses for communication.
- It is a circuit board installed in a computer that provides a dedicated network connection to the computer. It is also called network interface controller, network adapter, or LAN adapter.
- NIC is used to convert data into a digital signal.
- In the OSI model, NIC uses the physical layer to transmit signals and the network layer to transmit data packets.
- NIC offers both wired (using cables) and wireless (using Wi-Fi) data communication techniques.

b) Diagram:

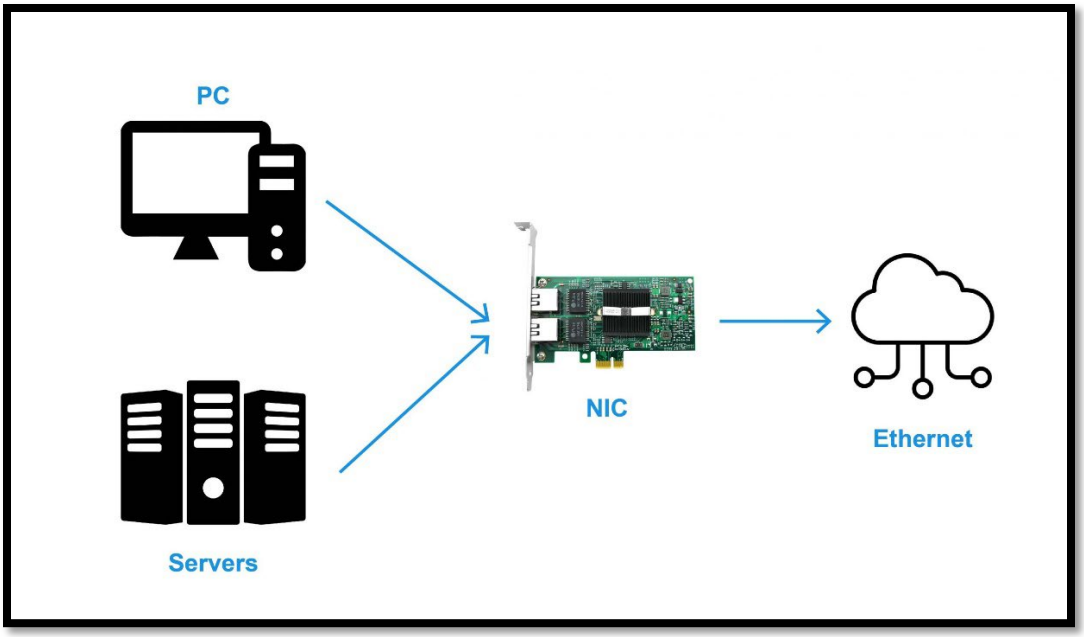


Fig: NIC in Networking.

c) Advantages:
- Provides dedicated connection to a network.
- Connects various devices to a network.

d) Disadvantages:
- May be more expensive.
- Requires installation and configuration.
- NIC is inconvenient as compared to the wireless card.
- NIC cards are not secure, so the data inside NIC is not safe

e) Limitations:
- Limited speed, operating at the speed of the device.
- For wired NIC, a hard-wired connection is required.

f) Technical Details:
- Operates at different speeds, e.g., 10/100/1000 Mbps or higher (10/25/40/100 Gbps).

B) Communication Channels

1) Wired Medium:

I. CAT5/CAT6 Cables:

a) Characteristics:
- Transmit data over twisted pairs of copper wires.
- They are also used to transmit signals over telephones and videos.
- They mostly connect using punch-down blocks and modular connectors.
- Most of the cables are unshielded; they do not have any additional insulation coating. They largely rely on the balanced line twisted pair design and differential signaling to ensure noise rejection.

b) Diagram:

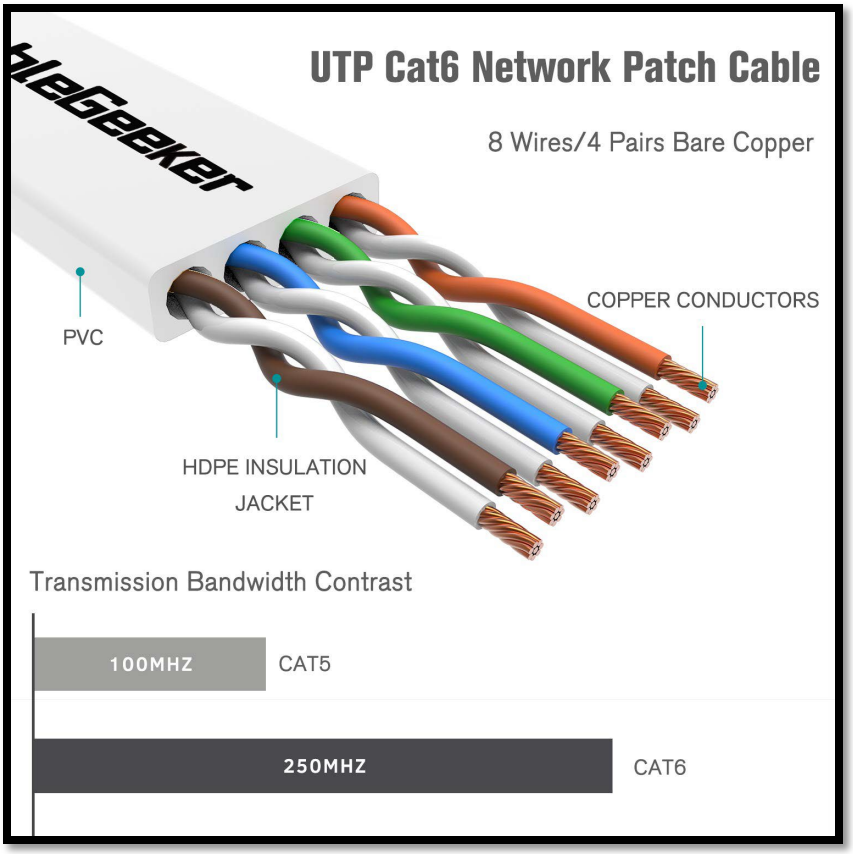


Fig: CAT6 cable.

c) Advantages:
- Reliable and fast data transmission.
- Inexpensive and widely available.

d) Disadvantages:
- Limited range (up to 100 meters).
- Susceptible to electromagnetic interference.

e) Limitations:
- Maximum length of 100 meters.

f) Technical Details:
- Support different speeds (10/100/1000 Mbps or 10/25/40/100 Gbps).
- Utilizes packet switching for data transmission.

II. Coaxial Cable:

a) Characteristics:
- Transmits data using a copper core surrounded by insulation and a braided shield.
- The core copper conductor is used for the transmission of signals and the insulator is used to provide insulation to the copper conductor and the insulator is surrounded by a braided metal conductor which helps to prevent the interference of electrical signals and prevent cross talk. This entire setup is again covered with a protective plastic layer to provide extra safety to the cable.

b) Diagram:

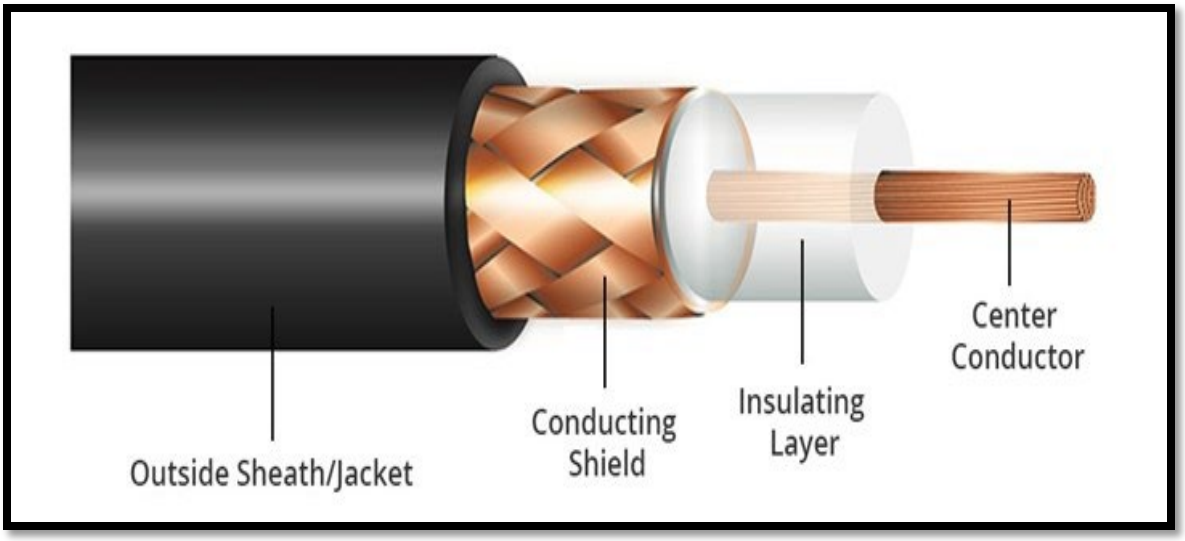


Fig: Coaxial Cable.

c) Structure of Coaxial Cable :
   - Copper conductor: A central conductor, which consists of copper. The conductor is the point at which data transmits.
   - Insulator: Dielectric plastic insulation around the copper conductor. It is used to maintain the spacing between the center conductor and shield.
   - Braided mesh: A braided mesh of copper helps to shield from electromagnetic interference, the braid provides a barrier against EMI moving into and out of the coaxial cable.
   - Protective plastic layer: An external polymer layer, which has a plastic coating. It is used to protect internal layers from damages.

d) Advantages:
   - Reliable and usable for longer distances than CAT5/CAT6.

e) Disadvantages:
   - More expensive than CAT5/CAT6.
   - Susceptible to electromagnetic interference.

f) Limitations:
   - Maximum length of 500 meters.

g) Technical Details:
   - Uses packet switching for data transmission.

III. Fiber Optic + Connectors:

a) Characteristics:
   - An Optical Fiber is a cylindrical fiber of glass which is hair thin size or any transparent dielectric medium.
   - The fiber which is used for optical communication is waveguides made of transparent dielectrics.
   - Transmits data using glass or plastic fibers and light signals.

b) Diagram:

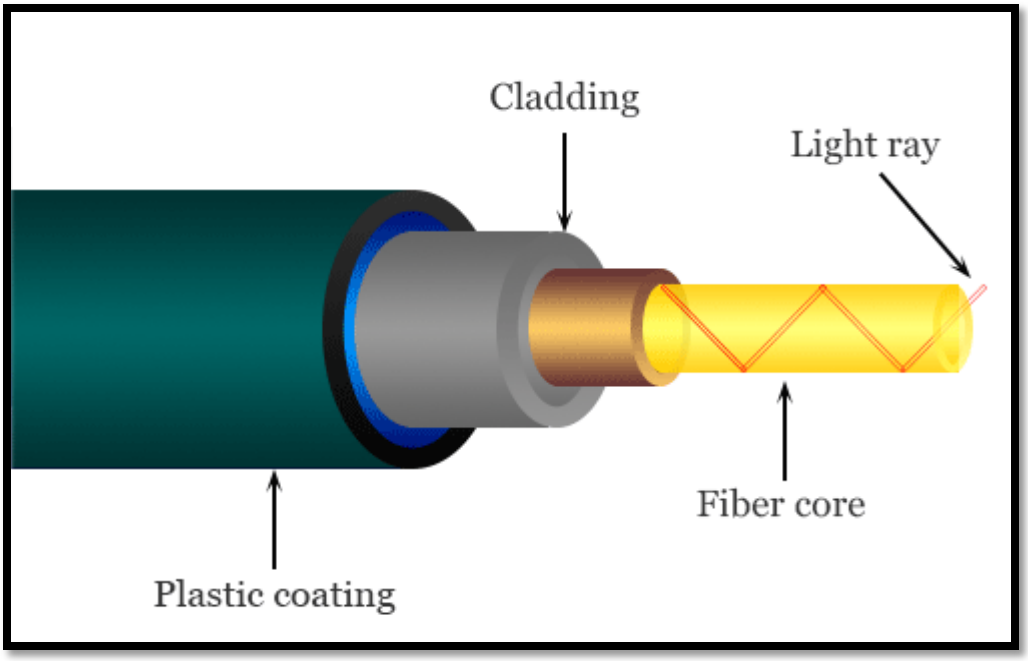


Fig: Optical Fiber Cable.

c) Main element of Fiber Optics :
- Core: It is the central tube of very thin size made of optically transparent dielectric medium and carries the light transmitter to receiver and the core diameter may vary from about 5um to 100 um.
- Cladding: It is outer optical material surrounding the core having reflecting index lower than core and cladding helps to keep the light within the core throughout the phenomena of total internal reflection.
- Buffer Coating: It is a plastic coating that protects the fiber made of silicon rubber. The typical diameter of the fiber after the coating is 250-300 um.

d) Advantages:
- Fast and secure data transmission.
- Suitable for longer distances than copper cables.

e) Disadvantages:
- More expensive than copper cables.
- Requires specialized equipment for installation and maintenance.

f) Limitations:
- Maximum length of several kilometers.

g) Technical Details:
- Uses packet switching for data transmission.

2) Wireless Medium:

I. Radio:

a) Radio waves can travel large distances as well as they are Susceptible to interference meaning they can penetrate any walls. (Omni-directional, these waves can move in all directions).

b) These are easy to generate and can penetrate through buildings. The requirement of radio waves is antennas, sending antennas where one can transmit its message and the other is receiving antennas.

c) The frequency range of radio waves: 3 KHz - 1GHz. Also, radio waves of frequency 300 KHz - 30 MHz can travel long distances.

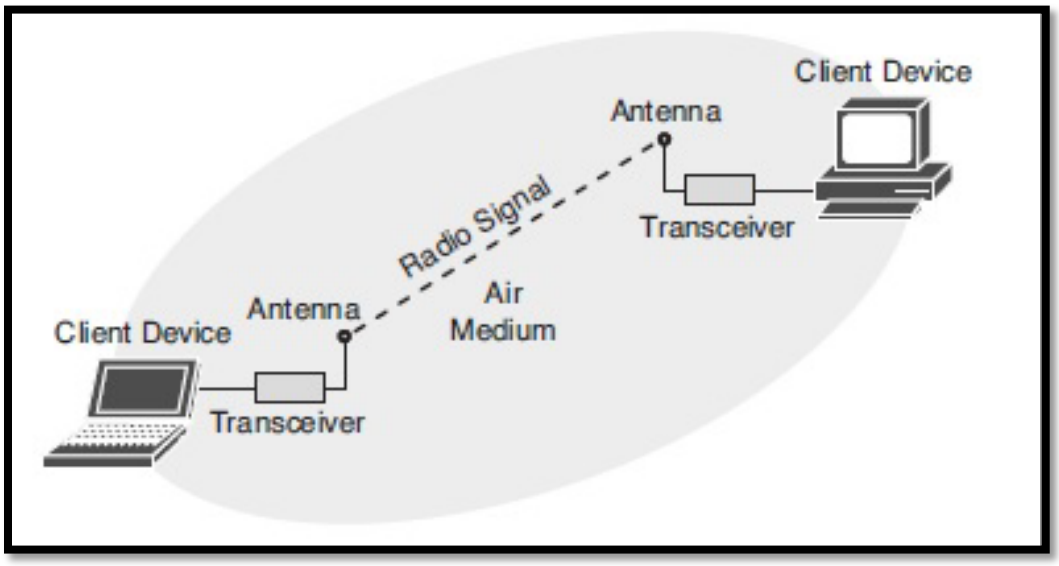


Fig: Radio Waves.

II. Microwaves:

a) Microwaves are a line of sight transmission, meaning both the antennas sending and receiving should be properly aligned.
b) The distance covered by the signal is directly proportional to the height of the antenna.
c) Microwaves have a frequency Range between 1GHz – 300GHz.
d) We used Microwaves in mobile phones communication and television distribution.
e) They are unidirectional, as they can move in only one direction, and therefore it is used in point-to-point communication or unicast communication such as radar and satellite.

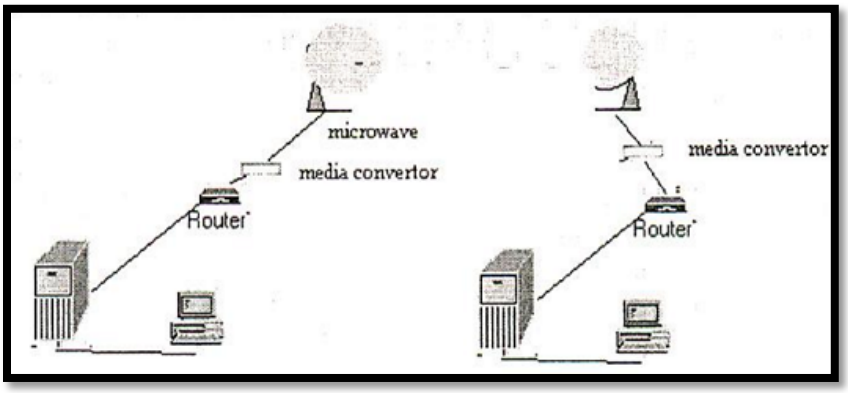


Fig: Microwaves in Networking.

III. Infrared:

a) Infrared is used for short-range communication like TV remotes, mobile phones, personal computers etc.
b) The Infrared is part of a spectrum that is not visible to the human eye.
c) The limitation of infrared rays is that they cannot penetrate any obstacles and can only use for short-range.
d) Infrared is used in night vision cameras as it has thermal properties. The frequency range of infrared rays 300GHz – 400THz.

e) They are used in TV remotes, PC devices like mice.

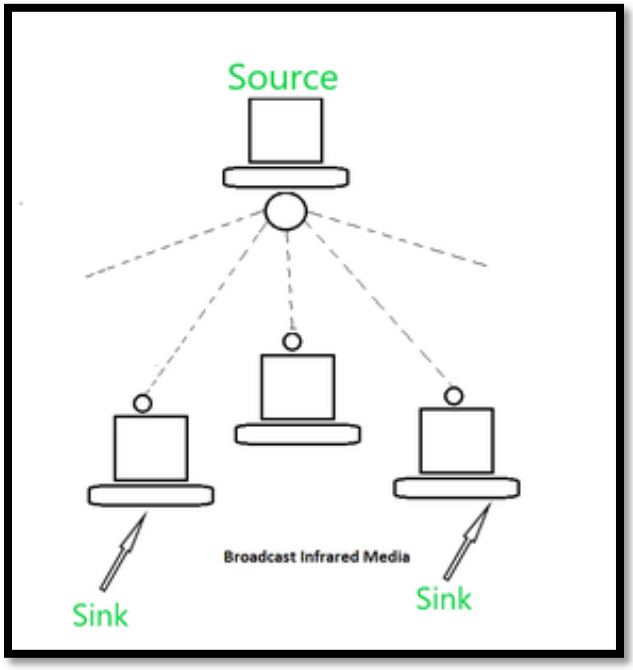


Fig: Infrared Waves.

IV. Frequency Band:

a) Used to allocate different frequencies for wireless communication.
b) Regulated by government agencies.

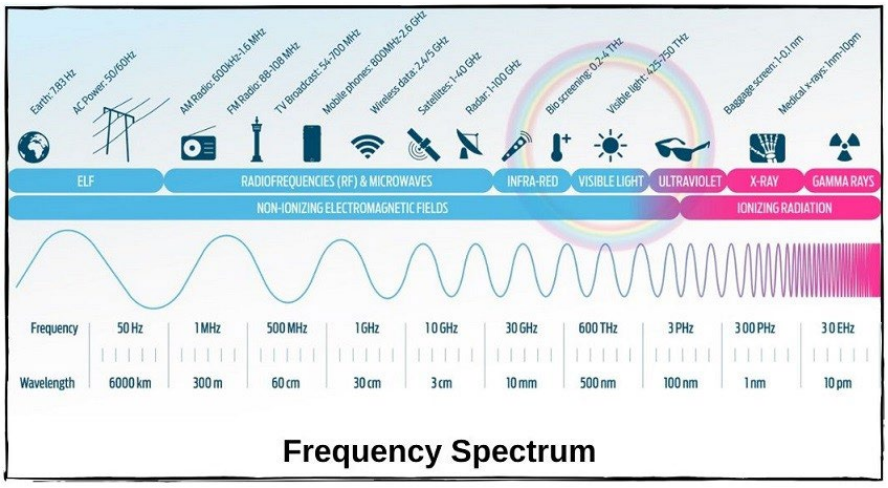


Fig: Frequency Spectrum

C) IP address (Classification of IP address, Sub netting, Super netting)

IP addresses are typically classified into two main categories: IPv4 and IPv6.

a. IPv4: •
    i. This is the most commonly used version of IP addresses.
    ii. It consists of 32 bits, typically represented in decimal format (e.g., 192.168.1.1).
    iii. IPv4 addresses are divided into several classes based on the first few bits of the address:
        1. Class A: Addresses in the range 1.0.0.0 to 126.0.0.0
        2. Class B: Addresses in the range 128.0.0.0 to 191.255.0.0
        3. Class C: Addresses in the range 192.0.0.0 to 223.255.255.0
        4. Class D: Reserved for multicast addressing
        5. Class E: Reserved for experimental use

b. IPv6:
    i. Introduced to address the limitations of IPv4, primarily the exhaustion of available addresses.
    ii. Uses 128 bits, providing a vastly larger address space.
    iii. Pv6 addresses are typically represented as eight groups of four hexadecimal digits, separated by colons (e.g., 2001:0db8:85a3:0000:0000:8a2e:0370:7334).

Subnetting:

a. Subnetting is the process of dividing a larger network into smaller subnetworks, allowing for efficient use of IP addresses and improved network performance.
b. Subnetting involves borrowing bits from the host portion of an IP address to create subnetworks. This process enables better organization and management of network resources.
c. Subnetting is often used to:
    i. Reduce network congestion
    ii. Improve network security
    iii. Optimize network performance
    iv. Allocate IP addresses efficiently

Supernetting:

a. Supernetting, also known as route aggregation or route summarization, is the opposite of subnetting.
b. It involves combining multiple contiguous network addresses into a single larger network. This process helps reduce the size of routing tables in routers and simplifies routing processes.
c. Supernetting is commonly used in large-scale networks to:
    i. Decrease routing table size • Improve routing efficiency
    ii. Minimize the number of entries in routing tables
    iii. Optimize network performance

Practical 2

Name of Program: Implement the Client-Server communication.

i. TCP communication.
ii. UDP communication.

Code:

i. TCP Communication:

a. TCP Client Code:

```
package TCP_Connection;

import java.io.*;
import java.net.*;

public class TCPClient {
   private Socket socket = null;
   private DataInputStream input = null;
   private DataOutputStream out = null;

   // constructor to put ip address and port
   public TCPClient(String address, int port) {
      // establish a connection
      try {
         socket = new Socket(address, port);
         System.out.println("Connected");

         // takes input from terminal
         input = new DataInputStream(System.in);

         // sends output to the socket
         out = new DataOutputStream(socket.getOutputStream());
      } catch (UnknownHostException u) {
         System.out.println(u);
         return;
      } catch (IOException i) {
         System.out.println(i);
         return;
      }

      // string to read message from input
      String line = "";
      String anLine = "";//

      // keep reading until "Over" is input
```

```
        while (true) {
            try {
                //SEND to Server
                System.out.printf(">> ");
                line = input.readLine();
                out.writeUTF(line);
                if (line.equals("END")) {//
                    break;//
                }//

                //READ From Server
                DataInputStream in = new DataInputStream(new
BufferedInputStream(socket.getInputStream()));//
                anLine = in.readUTF();//
                System.out.println("Server>> "+anLine);//
                if (anLine.equals("END")) {//
                    break;//
                }//

            } catch (IOException i) {
                System.out.println(i);
            }
        }

        System.out.println("-----Connection Ended-----");
        // close the connection
        try {
            input.close();
            out.close();
            socket.close();
        } catch (IOException i) {
            System.out.println(i);
        }
    }

    public static void main(String args[]) {
        TCPClient client = new TCPClient("127.0.0.1", 5000);
    }
}
```

b. TCP Server Code:

```
package TCP_Connection;

import java.net.*;
import java.io.*;

public class TCPServer {
   // initialize socket and input stream
   private Socket socket = null;
   private ServerSocket server = null;
   private DataInputStream in = null;


   // constructor with port
   public TCPServer(int port) {
      // starts server and waits for a connection
      try {
         server = new ServerSocket(port);
         System.out.println("Server started");
         System.out.println("Waiting for a client ...");

         socket = server.accept();
         System.out.println("Client accepted");

         // takes input from the client socket
         in = new DataInputStream(new
BufferedInputStream(socket.getInputStream()));

         //Initializing another input and out to send Messages
         DataInputStream input = new DataInputStream(System.in); //
         DataOutputStream out = new
DataOutputStream(socket.getOutputStream());;//

         String line = "";
         String anLine = "";//

         // reads message from client until "END" is sent
         while (true) {
            try {
               //READ From Client
               line = in.readUTF();
               System.out.println("Client>> " + line);
               if (line.equals("END")) {//
                  break;//
               }//
```

```
            //SEND to Client
            System.out.printf(">> ");//
            anLine=input.readLine();//
            out.writeUTF(anLine);//
            if (anLine.equals("END")) {//
               break;//
            }//

         } catch (IOException i) {
            System.out.println(i);
         }
      }
      System.out.println("-----Closing connection-----");

      // close connection
      socket.close();
      in.close();
   } catch (IOException i) {
      System.out.println(i);
   }
}

public static void main(String args[]) {
   TCPServer server = new TCPServer(5000);
}
}
```

Output:

ii. UDP Connection:

a. UDP Client Code:

```
import java.io.IOException;
import java.net.DatagramSocket;
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.util.Scanner;

public class UdpConnectionClient{
   public static void main(String[] args) throws IOException{
      Scanner sc = new Scanner(System.in);

      DatagramSocket dataSoc = new DatagramSocket();
      InetAddress ip = InetAddress.getLocalHost();

      byte buff[] = null;

      while(true){
         System.out.println(">> ");
         String input=sc.nextLine();
         buff=input.getBytes();
         DatagramPacket dpSend= new DatagramPacket(buff,buff.length,ip,5000);
         dataSoc.send(dpSend);

         if(input.equals("bye")){
                  System.out.println("Exiting");
                  break;
         }

         byte[] recievedData = new byte[123456];
         DatagramPacket dpRecieved =new DatagramPacket(recievedData,
recievedData.length);
         dataSoc.receive(dpRecieved);

         System.out.println("Server >> "+ dataToString(recievedData));
         if (dataToString(recievedData).toString().equals("bye")) {
            System.out.println("Exiting");
            break;
         }

      }
      dataSoc.close();
      sc.close();
   }
   public static StringBuilder dataToString(byte[] byteArray){
```

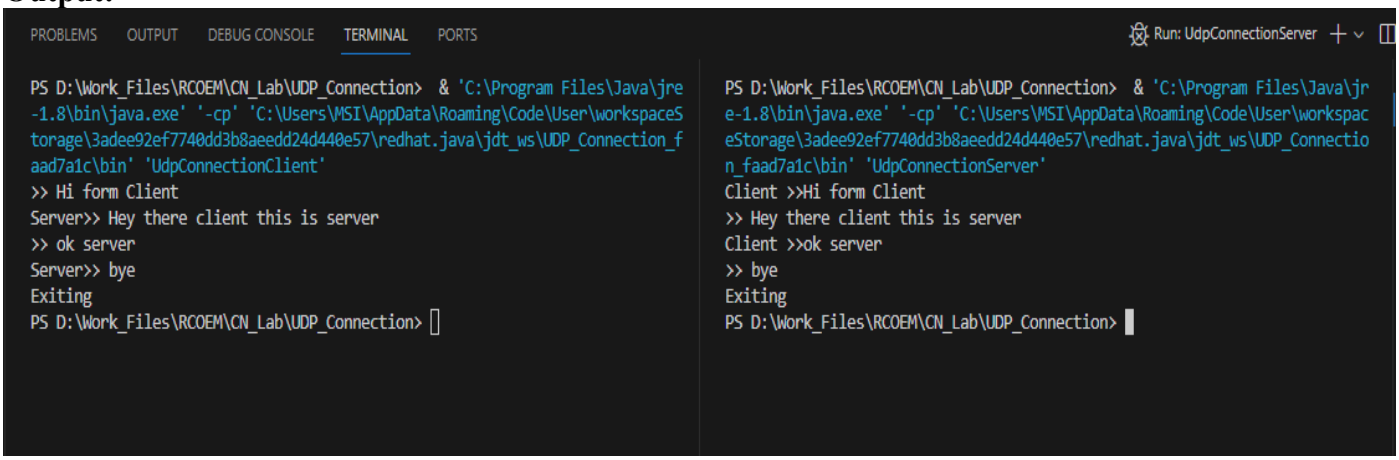
```
        if (byteArray == null) {
            return null;
        }
        StringBuilder res = new StringBuilder();
        int i = 0;
        while (byteArray[i]!=0) {
            res.append((char) byteArray[i]);
            i++;
        }
        return res;
    }
}
```

b. UDP Server Code:

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.io.IOException;
import java.net.SocketException;
import java.util.Scanner;
import java.net.InetAddress;

public class UdpConnectionServer extends Thread{
    public static void main(String[] args) throws IOException,SocketException{
        DatagramSocket ds= new DatagramSocket(5000);
        byte[] recievedData = new byte[123456];

        DatagramPacket dpRecieved = null;
        Scanner sc= new Scanner(System.in);

        while(true){
            dpRecieved =new DatagramPacket(recievedData, recievedData.length);
            ds.receive(dpRecieved);

            System.out.println("Client >> "+ dataToString(recievedData));
            if (dataToString(recievedData).toString().equals("bye")) {
                System.out.println("Exiting");
                break;
            }

            recievedData = new byte[123456];

            InetAddress ip = InetAddress.getLocalHost();
            System.out.println(">> ");
            String input=sc.nextLine();
```

```
            byte[] buff=input.getBytes();
            DatagramPacket dpSend = new
DatagramPacket(buff,buff.length,ip,dpRecieved.getPort());
            ds.send(dpSend);
            if (input.equals("bye")) {
                  System.out.println("Exiting");
                  break;
            }

      }
      sc.close();
      ds.close();
   }

   public static StringBuilder dataToString(byte[] byteArray){
      if (byteArray == null) {
         return null;
      }
      StringBuilder res = new StringBuilder();
      int i = 0;
      while (byteArray[i]!=0) {
         res.append((char) byteArray[i]);
         i++;
      }
      return res;
   }
}
```

Output:



Practical 3

Name of Program: Implementation of various framing techniques using client server communication

i. Byte Stuffing
ii. Bit Stuffing

Code:

i. Byte Stuffing:

TCPClientByteStuffing:

```
import java.io.*;
import java.net.*;
import java.util.ArrayList;
import java.util.List;

public class TCPClientByteStuffing {
   private Socket socket = null;
   private DataInputStream input = null;
   private DataOutputStream out = null;

   // constructor to put ip address and port
   public TCPClientByteStuffing(String address, int port) {
      // establish a connection
      try {
         socket = new Socket(address, port);
         System.out.println("Connected");
         // takes input from terminal
         input = new DataInputStream(System.in);
         // sends output to the socket
         out = new DataOutputStream(socket.getOutputStream());
      } catch (UnknownHostException u) {
         System.out.println(u);
         return;
      } catch (IOException i) {
         System.out.println(i);
         return;
      }
      String line = "";
      String stuffedLine = "";
      // String anLine = "";//

      while (true) {
         try {
            // SEND to Server
            System.out.printf(">> ");
            line = input.readLine();
```

```
            if (line.equals("END") || line.equals("/")) {//
               out.writeUTF(line);//
            } //
            else {

               List<String> sendArray = inputToArray(line);
               // stuffedLine = byteStuffing(line);
               for (int i = 0; i < sendArray.size(); i++) {
                  String iStr = sendArray.get(i);
                  out.writeUTF(iStr);
               }
            }
            if (line.equals("END")) {//
               break;//
            } //

         } catch (IOException i) {
            System.out.println(i);
         }
      }
      System.out.println("-----Connection Ended-----");
      // close the connection
      try {
         input.close();
         out.close();
         socket.close();
      } catch (IOException i) {
         System.out.println(i);
      }
   }

   public String byteStuffing(String inputData) {
      String data = inputData;
      String res = new String();

      // Data in each frame is stuffed by 'F' at beginning and end
      data = "F" + data + "F";
      for (int i = 0; i < data.length(); i++) {
         // Stuff with 'E' if 'F' is found in the data to be sent
         if (data.charAt(i) == 'F' && i != 0 && i != (data.length() - 1))
            res = res + 'E' + data.charAt(i);
         // Stuff with 'E' if 'E' is found in the data to be sent
         else if (data.charAt(i) == 'E')
            res = res + 'E' + data.charAt(i);
         else
```

```
            res = res + data.charAt(i);
        }
        return res;
    }

    public List<String> inputToArray(String inputString) {
        List<String> strArray = new ArrayList<String>();
        int framesNumber = (int) (inputString.length() / 6);
        int rem = (inputString.length() % 6);
        if (rem > 0) {
            framesNumber += 1;
        }

        int index = 0;
        while (index < inputString.length()) {
            strArray.add(inputString.substring(index, Math.min(index + 6, inputString.length())));
            index += 6;
        }
        int lastIndex = framesNumber - 1;
        if (strArray.get(lastIndex).length() < 6) {
            String str = strArray.get(lastIndex);
            for (int i = 0; i < 6 - rem; i++) {
                str = str + '\0';
            }
            strArray.set(lastIndex, str);
        }

        for (int i = 0; i < framesNumber; i++) {
            strArray.set(i, byteStuffing(strArray.get(i)));
        }
        System.out.println("Byte Stuffed Data :" + strArray);
        return strArray;
    }

    public static void main(String args[]) {
        TCPClientByteStuffing client = new TCPClientByteStuffing("127.0.0.1", 5000);
    }
}
```

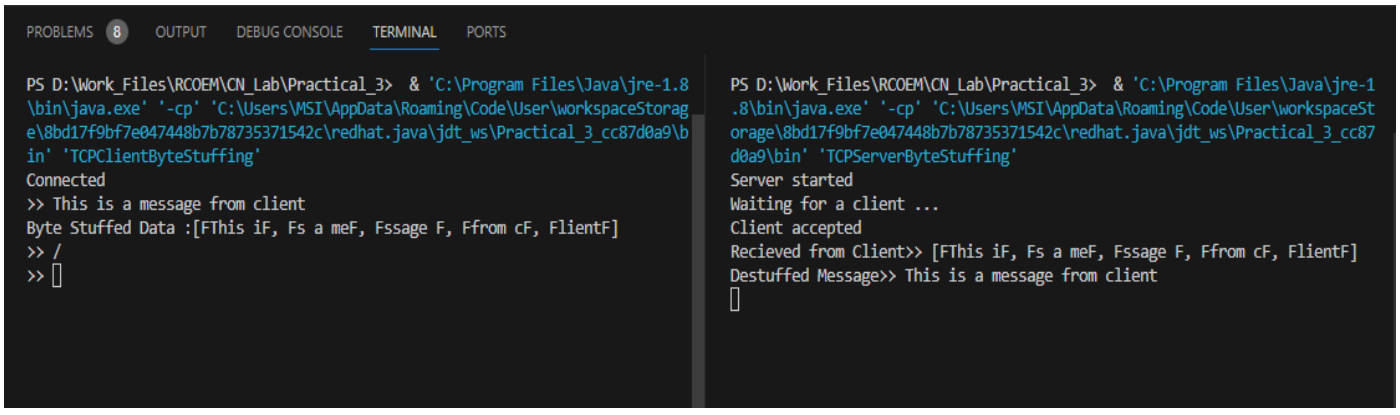
TCPServerByteStuffing:

```
import java.net.*;
import java.util.ArrayList;
import java.util.List;
// import java.util.Scanner;
import java.io.*;

public class TCPServerByteStuffing {
   // initialize socket and input stream
   private Socket socket = null;
   private ServerSocket server = null;
   private DataInputStream in = null;

   // constructor with port
   public TCPServerByteStuffing(int port) {
      // starts server and waits for a connection
      try {
         server = new ServerSocket(port);
         System.out.println("Server started");
         System.out.println("Waiting for a client ...");
         socket = server.accept();
         System.out.println("Client accepted");
         // takes input from the client socket
         in = new DataInputStream(new BufferedInputStream(socket.getInputStream()));

         List<String> lineArray = new ArrayList<String>();
         String line = "";//

         // reads message from client until "END" is sent
         while (true) {
            try {
               // READ From Client
               Boolean flag = true;
               do {
                  line = in.readUTF();
                  if (line.equals("/")) {
                     flag = false;
                     break;
                  }

                  if (line.equals("END")) {
                     flag = false;
                     lineArray.add(line);
                     break;
                  }
```

```
                lineArray.add(line);
            } while (flag);
            System.out.println("Recieved from Client>> " + lineArray);
            handleClientMessage(lineArray);
            if (lineArray.get(lineArray.size() - 1).equals("END")) {//
               break;//
            } //
            lineArray.clear();
         } catch (IOException e) {
            System.out.println(e.getMessage());
         }
      }
      System.out.println("-----Closing connection-----");
      // close connection
      socket.close();
      in.close();
   } catch (IOException i) {
      System.out.println(i);
   }
}

public String byteDestuffing(String inputData) {
   String data = inputData;
   StringBuilder res = new StringBuilder();

   for (int i = 0; i < data.length(); i++) {
      // If 'E' is encountered, skip it and check the next character
      if (data.charAt(i) == 'E') {
         i++;
         // If 'F' is encountered after 'E', then add 'F' to the result
         if (i < data.length() && data.charAt(i) == 'F') {
            res.append('F');
         }
         // If 'E' is encountered after 'E', then add 'E' to the result
         else if (i < data.length() && data.charAt(i) == 'E') {
            res.append('E');
         }
      }
      // If 'F' is encountered, skip it
      else if (data.charAt(i) == 'F') {
         continue;
      }
      // Otherwise, add the character to the result
      else {
         res.append(data.charAt(i));
      }
```

```
        }
        return res.toString();
    }

    public void handleClientMessage(List<String> messages) {
        List<String> destuffedMessages = new ArrayList<>();
        for (String message : messages) {
            destuffedMessages.add(byteDestuffing(message));
        }
        System.out.println("Destuffed Message>> " + elemToStr(destuffedMessages));
    }

    public String elemToStr(List<String> destuffedMessages) {
        StringBuilder result = new StringBuilder();
        for (String str : destuffedMessages) {
            result.append(str); // Append each string
        }

        // Remove the last space
        if (result.length() > 0) {
            result.setLength(result.length() - 1);
        }

        // Print the result
        return result.toString();
    }

    public static void main(String args[]) {
        TCPServerByteStuffing server = new TCPServerByteStuffing(5000);
    }
}
```

Output:



ii. Bits Stuffing:

TCPClientBitStuffing:

```
import java.io.*;
import java.net.*;
import java.util.ArrayList;
import java.util.List;

public class TCPClientBitStuffing {
   private Socket socket = null;
   private DataInputStream input = null;
   private DataOutputStream out = null;

   // constructor to put ip address and port
   public TCPClientBitStuffing(String address, int port) {
      // establish a connection
      try {
         socket = new Socket(address, port);
         System.out.println("Connected");
         // takes input from terminal
         input = new DataInputStream(System.in);
         // sends output to the socket
         out = new DataOutputStream(socket.getOutputStream());
      } catch (UnknownHostException u) {
         System.out.println(u);
         return;
      } catch (IOException i) {
         System.out.println(i);
         return;
      }
      String line = "";
      String stuffedLine = "";

      while (true) {
         try {
            // SEND to Server
            System.out.printf("Enter data(in bits) >> ");
            line = input.readLine();

            if (line.equals("END") || line.equals("/")) {//
               out.writeUTF(line);//
            } //
            else {

               List<String> sendArray = inputToArray(line);
               // stuffedLine = byteStuffing(line);
```

```
            for (int i = 0; i < sendArray.size(); i++) {
               String iStr = sendArray.get(i);
               out.writeUTF(iStr);
            }
         }
         if (line.equals("END")) {//
            break;//
         } //
      } catch (IOException i) {
         System.out.println(i);
      }
   }
   System.out.println("-----Connection Ended-----");
   // close the connection
   try {
      input.close();
      out.close();
      socket.close();
   } catch (IOException i) {
      System.out.println(i);
   }
}

public String bitStuffing(String inputData) {
   // Takes input of unstuffed data from user
   String data = inputData;
   int count = 0;
   String resultString = "";
   for (int i = 0; i < data.length(); i++) {
      char ch = data.charAt(i);
      if (ch == '1') {
         // count number of consecutive 1'resultString
         // in user'resultString data
         count++;
         if (count < 5)
            resultString += ch;
         else {
            // add one '0' after 5 consecutive 1'resultString
            resultString = resultString + ch + '0';
            count = 0;
         }
      } else {
         resultString += ch;
         count = 0;
      }
   }
```

```
        // add flag byte in the beginning
        // and end of stuffed data
        String flag = "01111110";
        resultString = flag + resultString + flag;
        return resultString;
    }

    public List<String> inputToArray(String inputString) {
        List<String> strArray = new ArrayList<String>();
        int framesNumber = (int) (inputString.length() / 6);
        int rem = (inputString.length() % 6);
        if (rem > 0) {
            framesNumber += 1;
        }

        int index = 0;
        while (index < inputString.length()) {
            strArray.add(inputString.substring(index, Math.min(index + 6, inputString.length())));
            index += 6;
        }
        int lastIndex = framesNumber - 1;
        System.out.println(strArray.get(lastIndex));
        if (strArray.get(lastIndex).length() < 6) {
            String str = strArray.get(lastIndex);
            for (int i = 0; i < 6 - rem; i++) {
                str = str + '\0';
            }
            strArray.set(lastIndex, str);
        }
        System.out.println(strArray);

        for (int i = 0; i < framesNumber; i++) {
            strArray.set(i, bitStuffing(strArray.get(i)));
        }
        System.out.println("Bit Stuffed Data:" + strArray);
        return strArray;
    }

    public static void main(String args[]) {
        TCPClientBitStuffing client = new TCPClientBitStuffing("127.0.0.1", 5000);
    }
}
```

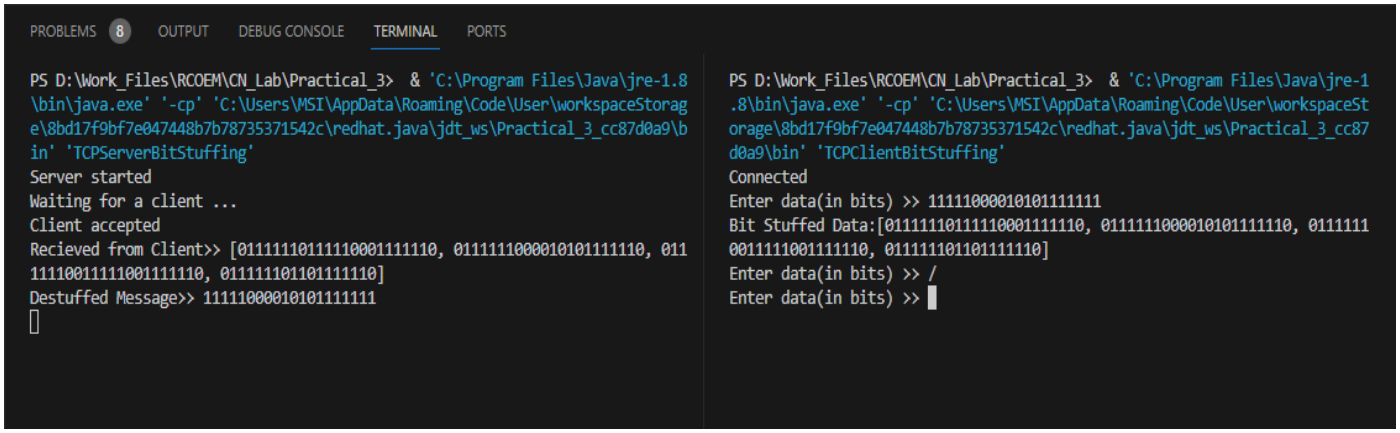
TCPServerBitStuffing:

```
import java.net.*;
import java.util.ArrayList;
import java.util.List;
import java.io.*;

public class TCPServerBitStuffing {
   // initialize socket and input stream
   private Socket socket = null;
   private ServerSocket server = null;
   private DataInputStream in = null;

   // constructor with port
   public TCPServerBitStuffing(int port) {
      // starts server and waits for a connection
      try {
         server = new ServerSocket(port);
         System.out.println("Server started");
         System.out.println("Waiting for a client ...");
         socket = server.accept();
         System.out.println("Client accepted");
         // takes input from the client socket
         in = new DataInputStream(new BufferedInputStream(socket.getInputStream()));

         List<String> lineArray = new ArrayList<String>();
         String line = "";//

         // reads message from client until "END" is sent
         while (true) {
            try {
               // READ From Client
               Boolean flag = true;
               do {
                  line = in.readUTF();
                  if (line.equals("/")) {
                     flag = false;
                     break;
                  }

                  if (line.equals("END")) {
                     flag = false;
                     lineArray.add(line);
                     break;
                  }
                  lineArray.add(line);
```

```
            } while (flag);
            System.out.println("Recieved from Client>> " + lineArray);
            handleClientMessage(lineArray);
            if (lineArray.get(lineArray.size() - 1).equals("END")) {//
               break;//
            } //
            lineArray.clear();
         } catch (IOException e) {
            System.out.println(e.getMessage());
         }
      }
      System.out.println("-----Closing connection-----");
      // close connection
      socket.close();
      in.close();
   } catch (IOException i) {
      System.out.println(i);
   }
}

public static String bitDestuffing(String stuffedData) {
   // Remove flag bytes
   stuffedData = stuffedData.substring(8, stuffedData.length() - 8);

   // Perform destuffing
   StringBuilder destuffedData = new StringBuilder();
   int count = 0;
   for (int i = 0; i < stuffedData.length(); i++) {
      char ch = stuffedData.charAt(i);
      if (ch == '1') {
         count++;
         destuffedData.append(ch);
         if (count == 5 && i != stuffedData.length() - 1) {
            i++;
            count = 0;
         }
      } else {
         count = 0;
         destuffedData.append(ch);
      }
   }
   return destuffedData.toString();
}

public void handleClientMessage(List<String> messages) {
   List<String> destuffedMessages = new ArrayList<>();
```

```
        for (String message : messages) {
            destuffedMessages.add(bitDestuffing(message));
        }
        System.out.println("Destuffed Message>> " + elemToStr(destuffedMessages));
    }

    public String elemToStr(List<String> destuffedMessages) {
        StringBuilder result = new StringBuilder();
        for (String str : destuffedMessages) {
            result.append(str); // Append each string
        }

        // Remove the last space
        if (result.length() > 0) {
            result.setLength(result.length() - 1);
        }

        // Print the result
        return result.toString();
    }

    public static void main(String args[]) {
        TCPServerBitStuffing server = new TCPServerBitStuffing(5000);
    }
}
```

Output:



Practical 4

Name of Program: Implementation of Error detection algorithm
iii. CRC (Cyclic Redundancy Check) Method.
iv. Hamming distance (code) method for detection/correction.

Code:

i. CRC (Cyclic Redundancy Check) Method :

TCPClient:

```
import java.io.*;
import java.net.*;
import java.util.ArrayList;
import java.util.List;

public class TCPClient {
   private Socket socket = null;
   private DataInputStream input = null;
   private DataOutputStream out = null;

   // constructor to put ip address and port
   public TCPClient(String address, int port) {
      // establish a connection
      try {
         socket = new Socket(address, port);
         System.out.println("Connected");
         // takes input from terminal
         input = new DataInputStream(System.in);
         // sends output to the socket
         out = new DataOutputStream(socket.getOutputStream());
      } catch (UnknownHostException u) {
         System.out.println(u);
         return;
      } catch (IOException i) {
         System.out.println(i);
         return;
      }
      String line = "";
      String stuffedLine = "";
      // String anLine = "";//

      while (true) {
         try {
            // SEND to Server
```

```
            System.out.printf(">> ");

            KeyValidityCheck KeyValChk = new KeyValidityCheck();
            String key = KeyValChk.key();
            DataValidityCheck dataValChk = new DataValidityCheck();
            String data = dataValChk.Data();
            out.writeUTF(CRC.EncodeData(data, key));
            out.writeUTF(key);
            if (line.equals("END")) {//
               break;//
            } //

         } catch (IOException i) {
            System.out.println(i);
         }
      }
      System.out.println("-----Connection Ended-----");
      // close the connection
      try {
         input.close();
         out.close();
         socket.close();
      } catch (IOException i) {
         System.out.println(i);
      }
   }

   public String byteStuffing(String inputData) {
      String data = inputData;
      String res = new String();

      // Data in each frame is stuffed by 'F' at beginning and end
      data = "F" + data + "F";
      for (int i = 0; i < data.length(); i++) {
         // Stuff with 'E' if 'F' is found in the data to be sent
         if (data.charAt(i) == 'F' && i != 0 && i != (data.length() - 1))
            res = res + 'E' + data.charAt(i);
         // Stuff with 'E' if 'E' is found in the data to be sent
         else if (data.charAt(i) == 'E')
            res = res + 'E' + data.charAt(i);
         else
            res = res + data.charAt(i);
      }
      return res;
   }
```

```
    public List<String> inputToArray(String inputString) {
        List<String> strArray = new ArrayList<String>();
        int framesNumber = (int) (inputString.length() / 6);
        int rem = (inputString.length() % 6);
        if (rem > 0) {
            framesNumber += 1;
        }

        int index = 0;
        while (index < inputString.length()) {
            strArray.add(inputString.substring(index, Math.min(index + 6, inputString.length())));
            index += 6;
        }
        int lastIndex=framesNumber-1;
        System.out.println(strArray.get(lastIndex));
        if (strArray.get(lastIndex).length() < 6) {
            String str=strArray.get(lastIndex);
            for (int i=0;i<6-rem;i++){
                str=str+'X';
            }
            strArray.set(lastIndex, str);
        }
        System.out.println(strArray);

        for (int i=0;i<framesNumber;i++) {
            strArray.set(i, byteStuffing(strArray.get(i)));
        }
        System.out.println(strArray);
        return strArray;
    }

    public static void main(String args[]) {
        TCPClient client = new TCPClient("127.0.0.1", 5000);
    }
}
```

TCPServer:

```
import java.net.*;
import java.io.*;

public class TCPServer {
   // initialize socket and input stream
   private Socket socket = null;
   private ServerSocket server = null;
   private DataInputStream in = null;

   // constructor with port
   public TCPServer(int port) {
      // starts server and waits for a connection
      try {
         server = new ServerSocket(port);
         System.out.println("Server started");
         System.out.println("Waiting for a client ...");
         socket = server.accept();
         System.out.println("Client accepted");
         // takes input from the client socket
         in = new DataInputStream(new BufferedInputStream(socket.getInputStream()));

         String data = "";

         // reads message from client until "END" is sent
         while (true) {
            try {
               // READ From Client
               data = in.readUTF();
               System.out.println("Client>> Data:" + data);
               String key;
               key = in.readUTF();
               System.out.println("Client>> Key:" + key);
               CRC.Receiver(data + CRC.Mod2Div(data + new String(new char[key.length() -
1]).replace("\0", "0"), key), key);
               if (data.equals("END")) {//
                  break;//
               } //
            } catch (IOException i) {
               System.out.println(i);
            }
         }
         System.out.println("-----Closing connection-----");
         // close connection
         socket.close();
```

```
            in.close();
        } catch (IOException i) {
            System.out.println(i);
        }
    }

    public static void main(String args[]) {
        TCPServer server = new TCPServer(5000);
    }
}
```

CRC:

```
public class CRC {
    // Returns XOR of 'a' and 'b'
    // (both of same length)
    public static String Xor(String a, String b) {
        // Initialize result
        String result = "";
        int n = b.length();
        // Traverse all bits, if bits are
        // same, then XOR is 0, else 1
        for (int i = 1; i < n; i++) {
            if (a.charAt(i) == b.charAt(i))
                result += "0";
            else
                result += "1";
        }
        return result;
    }

    // Performs Modulo-2 division
    static String Mod2Div(String dividend, String divisor) {
        // Number of bits to be XORed at a time.
        int pick = divisor.length();
        // Slicing the dividend to appropriate
        // length for particular step
        String tmp = dividend.substring(0, pick);
        int n = dividend.length();
        while (pick < n) {
            if (tmp.charAt(0) == '1')
                // Replace the dividend by the result
                // of XOR and pull 1 bit down
                tmp = Xor(divisor, tmp) + dividend.charAt(pick);
            else
                // If leftmost bit is '0'.
```

```
            // If the leftmost bit of the dividend (or
            // the part used in each step) is 0, the
            // step cannot use the regular divisor; we
            // need to use an all-0s divisor.
            tmp = Xor(new String(new char[pick]).replace("\0", "0"), tmp) + dividend.charAt(pick);
        // Increment pick to move further
        pick += 1;
    }
    // For the last n bits, we have to carry it out
    // normally as increased value of pick will cause
    // Index Out of Bounds.
    if (tmp.charAt(0) == '1')
        tmp = Xor(divisor, tmp);
    else
        tmp = Xor(new String(new char[pick]).replace("\0", "0"), tmp);
    return tmp;
}

// Function used at the sender side to encode
// data by appending remainder of modular division
// at the end of data.
static String EncodeData(String data, String key) {
    int l_key = key.length();
    // Appends n-1 zeroes at end of data
    String appended_data = (data + new String(new char[l_key - 1]).replace("\0", "0"));
    String remainder = Mod2Div(appended_data, key);
    // Append remainder in the original data
    String codeword = data + remainder;
    System.out.println("Remainder : " + remainder);
    System.out.println(
          "Encoded Data (Data + Remainder) :" + codeword
                + "\n");
    return codeword;
}

// checking if the message received by receiver is
// correct or not. If the remainder is all 0 then it is
// correct, else wrong.
static void Receiver(String data, String key) {
    String currxor = Mod2Div(data.substring(0, key.length()), key);
    int curr = key.length();
    while (curr != data.length()) {
        if (currxor.length() != key.length()) {
            currxor += data.charAt(curr++);
        } else {
            currxor = Mod2Div(currxor, key);
```

```
            }
        }
        if (currxor.length() == key.length()) {
            currxor = Mod2Div(currxor, key);
        }
        if (currxor.contains("1")) {
            System.out.println(
                    "there is some error in data");
        } else {
            System.out.println("correct message received");
        }
    }

    // Driver code
    public static void main(String[] args) {
        // try {
        //     System.out.println("\nSender side...");
        //     EncodeData(data, key);
        //     System.out.println("Receiver side...");
        //     Receiver(data + Mod2Div(data + new String(new char[key.length() - 1]).replace("\0", "0"),
key), key);
        // } catch (Exception e) {
        //     e.printStackTrace();
        // }
    }
}
```

DataValidityCheck:

```
import java.util.Scanner;

public class DataValidityCheck {
    public String Data() {
        Scanner dataScan = new Scanner(System.in);
        String data;
        Boolean isValid;
        do {
            System.out.println("Enter 8 bit message: ");
            if (dataScan.hasNextLine()) { // Check if there is a next line available
                data = dataScan.nextLine();
            } else {
                // Handle the case where no input is available
                System.out.println("No input found. Please try again.");
                data = ""; // Set data to an empty string or handle it based on your requirements
            }
```

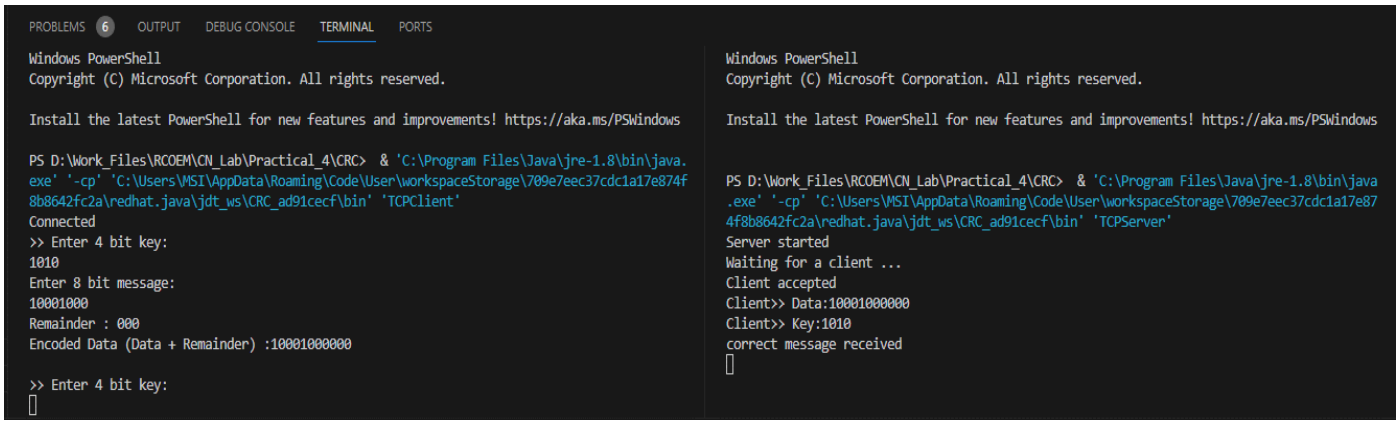
```
            isValid = true;
            if (data.length() != 8) {
                System.out.println("Data not entered Properly: Error Length not equal to 8");
                isValid = false;
            }
            for (int i = 0; i < data.length(); i++) {
                if (data.charAt(i) != '1' && data.charAt(i) != '0') {
                    isValid = false;
                    System.out.println("Data not entered Properly: Error wrong format: Enter Bits 1,0");
                    break;
                }
            }
        } while (!isValid);
        // dataScan.close();
        return data;
    }
}
```

KeyValidityCheck:

```
import java.util.Scanner;

public class KeyValidityCheck {
    public String key() {
        Scanner KeyScan = new Scanner(System.in);
        String key;
        Boolean isValid;
        do {
            System.out.println("Enter 4 bit key: ");
            key = KeyScan.nextLine();
            if (key.length() != 4) {
                System.out.println("Key not entered Properly: Error Length not equal to 4");
            }
            for (int i = 0; i < key.length(); i++) {
                if (i > 1 || i < 0) {
                    if (key.charAt(i) == '1') {
                        isValid = true;
                    } else if (key.charAt(i) == '0') {
                        isValid = true;
                    } else {
                        isValid = false;
                        System.out.println("Key not entered Properly: Error wrong format: Enter Bits 1,0");
                        break;
                    }
                }
            }
```

```
        } while (key.length() != 4);
        // KeyScan.close();
        return key;
    }
}
```

Output:



ii. Hamming distance (code) method for detection/correction:

TCPClient:

```
import java.io.*;
import java.net.*;

public class TCPClient {
   private Socket socket = null;
   private DataInputStream input = null;
   private DataOutputStream out = null;

   // constructor to put ip address and port
   public TCPClient(String address, int port) {
      // establish a connection
      try {
         socket = new Socket(address, port);
         System.out.println("Connected");
         // takes input from terminal
         input = new DataInputStream(System.in);
         // sends output to the socket
         out = new DataOutputStream(socket.getOutputStream());
      } catch (UnknownHostException u) {
         System.out.println(u);
         return;
      } catch (IOException i) {
         System.out.println(i);
         return;
      }
      String line = "";
      // String anLine = "";//

      while (true) {
         try {
            // SEND to Server
            System.out.printf(">> ");
            // line = input.readLine();
            // List<String> sendArray=inputToArray(line);
            // stuffedLine = byteStuffing(line);
            // for (int i = 0; i < sendArray.size(); i++) {
            //    String iStr=sendArray.get(i);
            //    out.writeUTF(iStr);
            // }
            // KeyValidityCheck KeyValChk = new KeyValidityCheck();
            // String key = KeyValChk.key();
```

```
            DataValidityCheck dataValChk = new DataValidityCheck();
            String data = dataValChk.Data();
            String hammingCode= HammingCodeGenerator.generateHammingCode(data);
            out.writeUTF(hammingCode);
            if (line.equals("END")) {//
               break;//
            } //

            // // READ From Server
            // DataInputStream in = new DataInputStream(new
            // BufferedInputStream(socket.getInputStream()));//
            // anLine = in.readUTF();//
            // System.out.println("Server>> " + anLine);//
            // if (anLine.equals("END")) {//
            // break;//
            // } //

         } catch (IOException i) {
            System.out.println(i);
         }
      }
      System.out.println("-----Connection Ended-----");
      // close the connection
      try {
         input.close();
         out.close();
         socket.close();
      } catch (IOException i) {
         System.out.println(i);
      }
   }

   public static void main(String args[]) {
      TCPClient client = new TCPClient("127.0.0.1", 5000);
   }
}
```

TCPServer:

```
import java.net.*;
// import java.util.Scanner;
import java.io.*;

public class TCPServer {
   // initialize socket and input stream
   private Socket socket = null;
   private ServerSocket server = null;
   private DataInputStream in = null;

   // constructor with port
   public TCPServer(int port) {
      // starts server and waits for a connection
      try {
         server = new ServerSocket(port);
         System.out.println("Server started");
         System.out.println("Waiting for a client ...");
         socket = server.accept();
         System.out.println("Client accepted");
         // takes input from the client socket
         in = new DataInputStream(new BufferedInputStream(socket.getInputStream()));

         // Initializing another input and out to send Messages
         // DataInputStream input = new DataInputStream(System.in); //
         // DataOutputStream out = new DataOutputStream(socket.getOutputStream());
         ;//
         String data = "";
         // String anLine = "";//

         // reads message from client until "END" is sent
         while (true) {
            try {
               // READ From Client
               data = in.readUTF();
               System.out.println("Client>> " + data);
               int hammingCode[]=HammingCodeCheckerCorrector.checkAndCorrect(data);
               System.out.println("Decoded Data: " +
HammingCodeCheckerCorrector.decodeHammingCode(hammingCode));
               if (data.equals("END")) {//
                  break;//
               } //
               //   // SEND to Client
               // System.out.printf(">> ");//
               // anLine = input.readLine();//
```

```
                // out.writeUTF(anLine);//
                // if (anLine.equals("END")) {//
                //     break;//
                // } //
            } catch (IOException i) {
                System.out.println(i);
            }
        }
        System.out.println("-----Closing connection-----");
        // close connection
        socket.close();
        in.close();
    } catch (IOException i) {
        System.out.println(i);
    }
  }

  public static void main(String args[]) {
    TCPServer server = new TCPServer(5000);
  }
}
```

HammingCodeGenerator:

```
public class HammingCodeGenerator {
  public static String generateHammingCode(String pureData) {
    int data[]=new int[pureData.length()];
    for (int i = 0; i < pureData.length(); i++) {
      data[i]=Character.getNumericValue(pureData.charAt(i));
    }
    int m = data.length; // Number of data bits
    int r = calculateParityBitCount(m); // Number of parity bits

    // Total number of bits in the Hamming code
    int n = m + r;

    // Create an array to store the Hamming code
    int[] hammingCode = new int[n];

    // Copy data bits to Hamming code, leaving space for parity bits
    int dataIndex = 0; // Index for iterating over data bits
    for (int i = 0; i < n; i++) {
      // Check if the current index is a power of 2 (parity bit position)
      if (!isPowerOfTwo(i + 1)) {
        // If not a parity bit position, copy data bit
        hammingCode[i] = data[dataIndex];
```

```
            dataIndex++;
        }
    }

    // Calculate parity bits
    for (int i = 0; i < r; i++) {
        int parityBitIndex = (int) Math.pow(2, i) - 1; // Index of parity bit

        // Calculate parity bit value
        hammingCode[parityBitIndex] = calculateParityBit(hammingCode, parityBitIndex);
    }

    StringBuilder hammingCodeString=new StringBuilder();
    for (int i = 0; i < hammingCode.length; i++) {
        hammingCodeString.append(hammingCode[i]);
    }
    return hammingCodeString.toString();
}

// Method to calculate the number of parity bits required for given number of data bits
public static int calculateParityBitCount(int dataBitCount) {
    int r = 0; // Number of parity bits

    // Calculate the number of parity bits required
    while (Math.pow(2, r) < (dataBitCount + r + 1)) {
        r++;
    }

    return r;
}

// Method to calculate the parity bit value
public static int calculateParityBit(int[] hammingCode, int parityBitIndex) {
    int parity = 0;
    int n = hammingCode.length;

    // Calculate the XOR of all bits covered by this parity bit
    for (int i = parityBitIndex; i < n; i += 2 * (parityBitIndex + 1)) {
        for (int j = i; j < Math.min(i + parityBitIndex + 1, n); j++) {
            parity ^= hammingCode[j];
        }
    }

    return parity;
}
```

```
    // Method to check if a number is a power of two
    public static boolean isPowerOfTwo(int num) {
        return num != 0 && (num & (num - 1)) == 0;
    }
}
```

HammingCodeCheckerCorrector:

```
public class HammingCodeCheckerCorrector {

    // Method to calculate the number of parity bits required
    public static int calculateParityBits(int dataSize) {
        int m = 0;
        while (Math.pow(2, m) < (dataSize + m + 1)) {
            m++;
        }
        return m;
    }

    // Method to calculate parity bit
    public static int calculateParity(int[] hammingCode, int parityIndex) {
        int parity = 0;
        for (int i = parityIndex; i < hammingCode.length; i++) {
            if (((i + 1) & (parityIndex + 1)) != 0) {
                parity ^= hammingCode[i];
            }
        }
        return parity;
    }

    // Method to check and correct Hamming code
    public static int[] checkAndCorrect(String data) {
        int received[]=new int[data.length()];
        for (int i = 0; i < data.length(); i++) {
            received[i]=Character.getNumericValue(data.charAt(i));
        }
        int parityBits = calculateParityBits(received.length);
        int[] errorPositions = new int[parityBits];
        int errorIndex = 0;
        for (int i = 0; i < parityBits; i++) {
            int parityIndex = (int) Math.pow(2, i) - 1;
            int parity = calculateParity(received, parityIndex);
            if (parity != 0) {
                errorPositions[errorIndex++] = parityIndex;
            }
        }
```

```
        if (errorIndex != 0) {
            int errorPosition = 0;
            for (int i = 0; i < errorIndex; i++) {
                errorPosition += errorPositions[i];
            }
            if (errorPosition < received.length) {
                received[errorPosition] ^= 1;
                System.out.println("Error detected and corrected at bit position " + errorPosition);
                return received;
            } else {
                System.out.println("Error detected but cannot be corrected");
            }
        } else {
            System.out.println("No error detected");
            return received;
        }
        return received;
    }

    public static String decodeHammingCode(int[] received) {
        int parityBits = calculateParityBits(received.length);
        int[] decodedData = new int[received.length - parityBits+1];
        int j = 0;
        for (int i = 0; i < received.length; i++) {
            if ((i + 1 & i) != 0) { // Skip parity bits
                // System.out.print(received[i]);
                decodedData[j++] = received[i];
            }
        }
        StringBuilder decodeDataStr=new StringBuilder();
        for (int i = 0; i < decodedData.length; i++) {
            decodeDataStr.append(decodedData[i]);
        }
        return decodeDataStr.toString();
    }
}
```

DataValidityCheck:

```
import java.util.Scanner;

public class DataValidityCheck {
    public String Data() {
        Scanner dataScan = new Scanner(System.in);
        String data;
        Boolean isValid;
```

```
        do {
            System.out.println("Enter 4 bit message: ");
            if (dataScan.hasNextLine()) { // Check if there is a next line available
                data = dataScan.nextLine();
            } else {
                // Handle the case where no input is available
                System.out.println("No input found. Please try again.");
                data = ""; // Set data to an empty string or handle it based on your requirements
            }
            isValid = true;
            if (data.length() != 4) {
                System.out.println("Data not entered Properly: Error Length not equal to 4");
                isValid = false;
            }
            for (int i = 0; i < data.length(); i++) {
                if (data.charAt(i) != '1' && data.charAt(i) != '0') {
                    isValid = false;
                    System.out.println("Data not entered Properly: Error wrong format: Enter Bits 1,0");
                    break;
                }
            }
        } while (!isValid);
        // dataScan.close();
        return data;
    }
}
```

Output: