Experiment 1

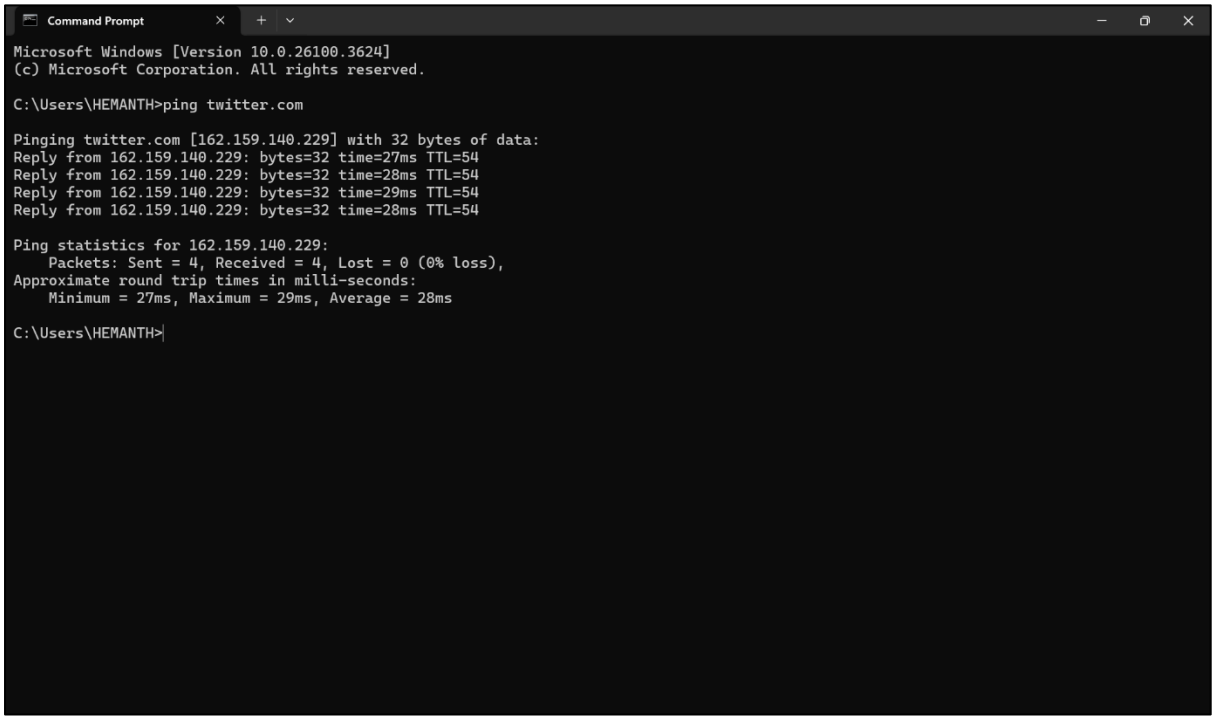
AIM: To study of basic network commands and network configuration commands

a. Ping
b. Tracent
c. Ipconfig
d. Hostname
e. Nslookup
f. netstat

ping: It is one of the basic networking commands to test the connection between the local machine and the host server. This command sends a small amount of data to the host server,and in return the host server sends a replay to the computer information like the ip address of the host server, the amount of data sent, time to live. And time needed for sending and the data are recorded and displayeded to the user.

Ex:

C:\Users\HEMANTH>ping twitter.com

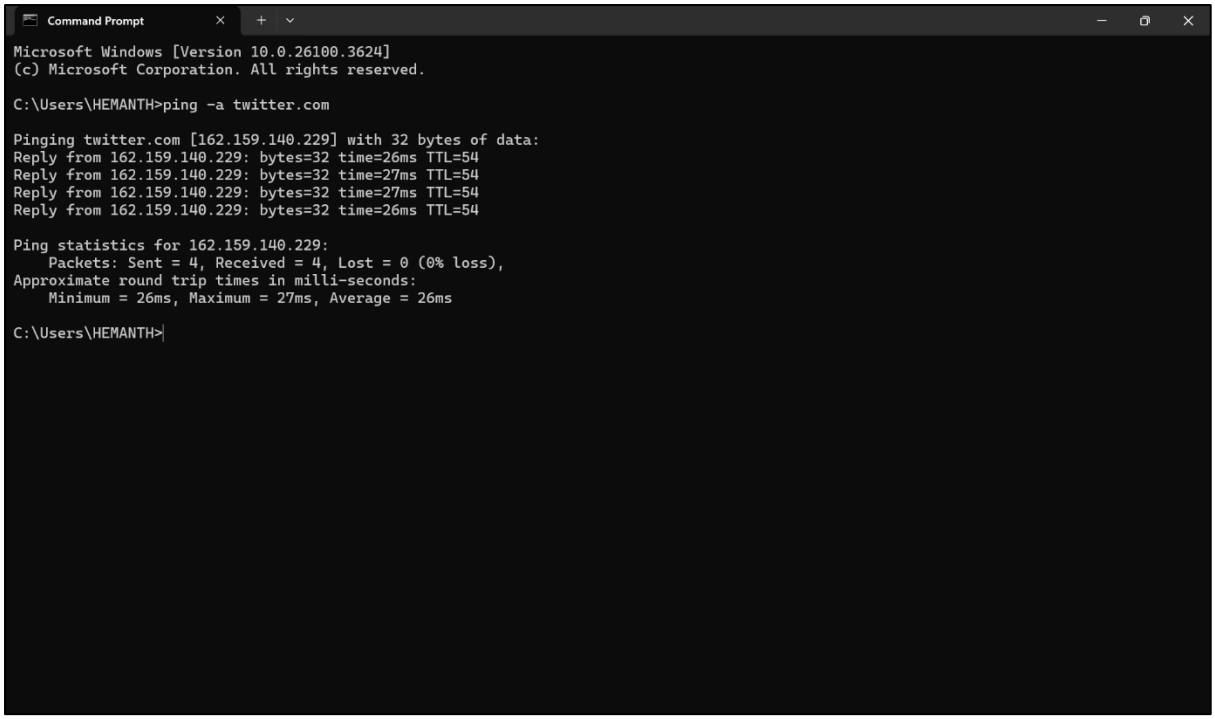


-a: The "-a" option resolves the hostname to the respective ip address

The command ping -a is used to perform a ping test while attempting the resolve the hostname of the target ip address.

Ex:

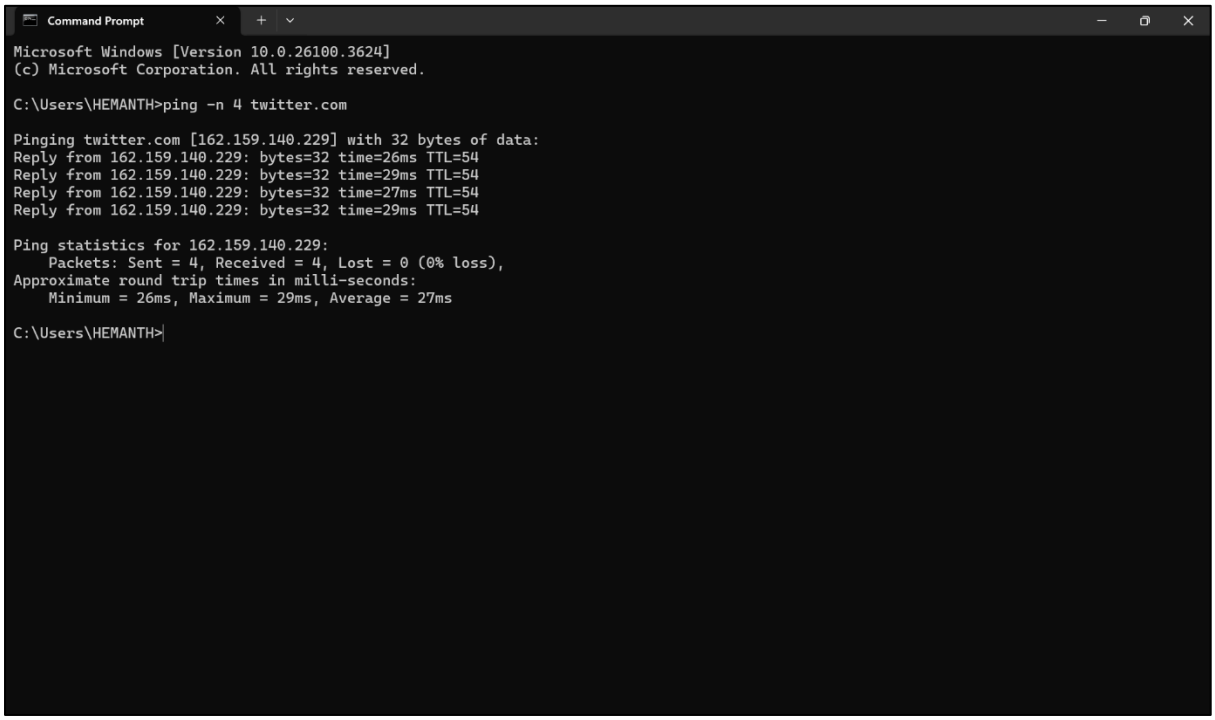
C:\Users\HEMANTH>ping -a twitter.com



-n count: The -n <count> option in the ping command specifies the number of echo request packets to send. This is useful when you want to control the number of ping attempts instead of running indefinitely.

Ex:

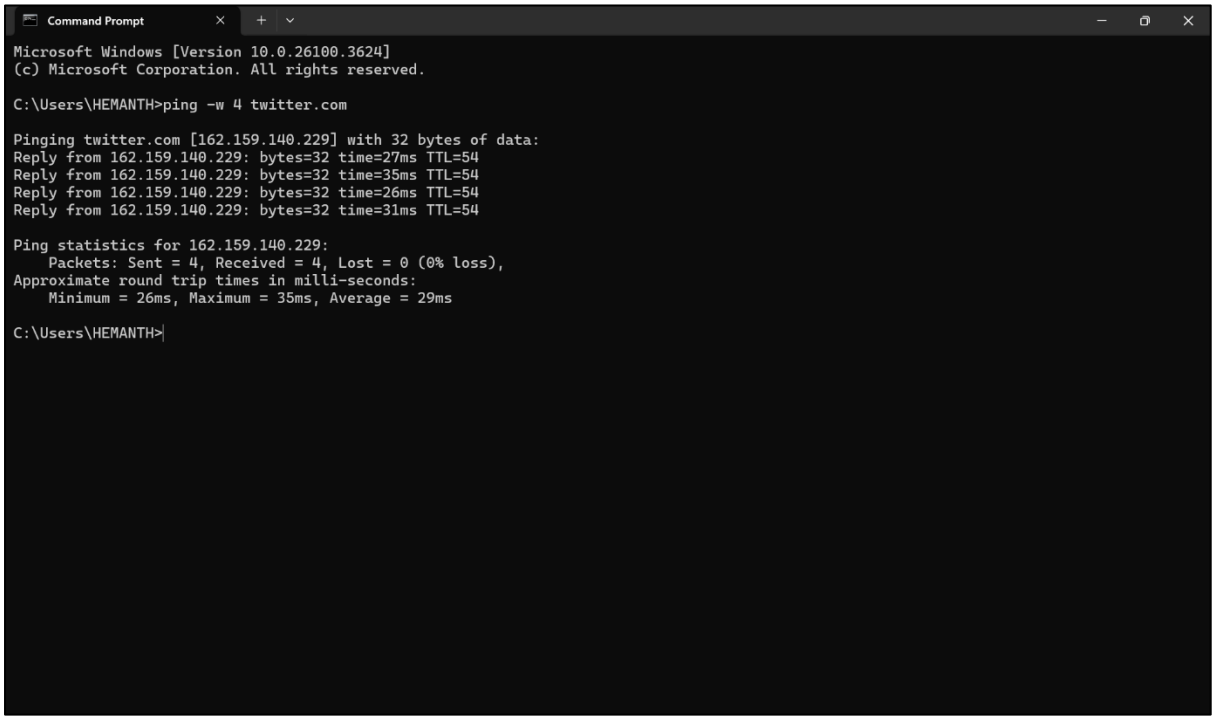
C:\Users\HEMANTH>ping -n 4 twitter.com



-w timeout: The -w <timeout> option in the ping command sets a timeout (in milliseconds) for each reply. If the response is not received within the specified time, the request is considered lost.

Ex:

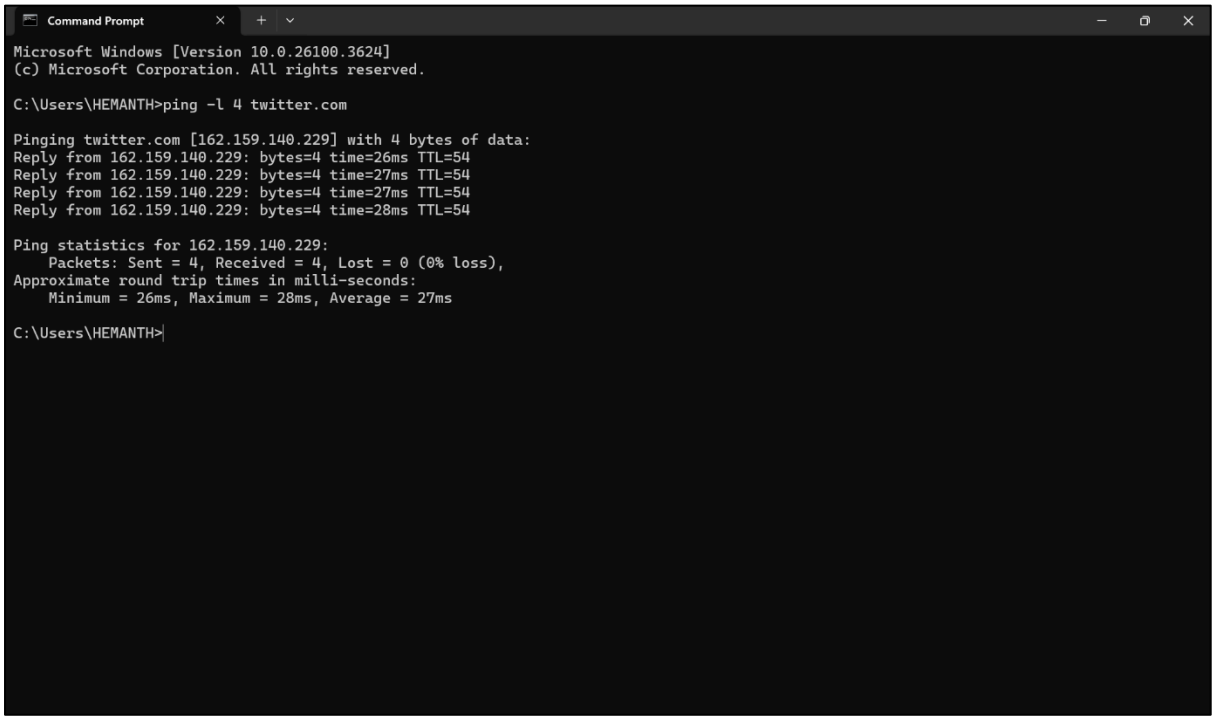
C:\Users\HEMANTH>ping -w 4 twitter.com



-l size: The -l <size> option in the ping command specifies the size of the packet (in bytes) to send. This is useful for testing network performance and identifying MTU (Maximum Transmission Unit) limits.

Ex:

C:\Users\HEMANTH>ping -l 4 twitter.com

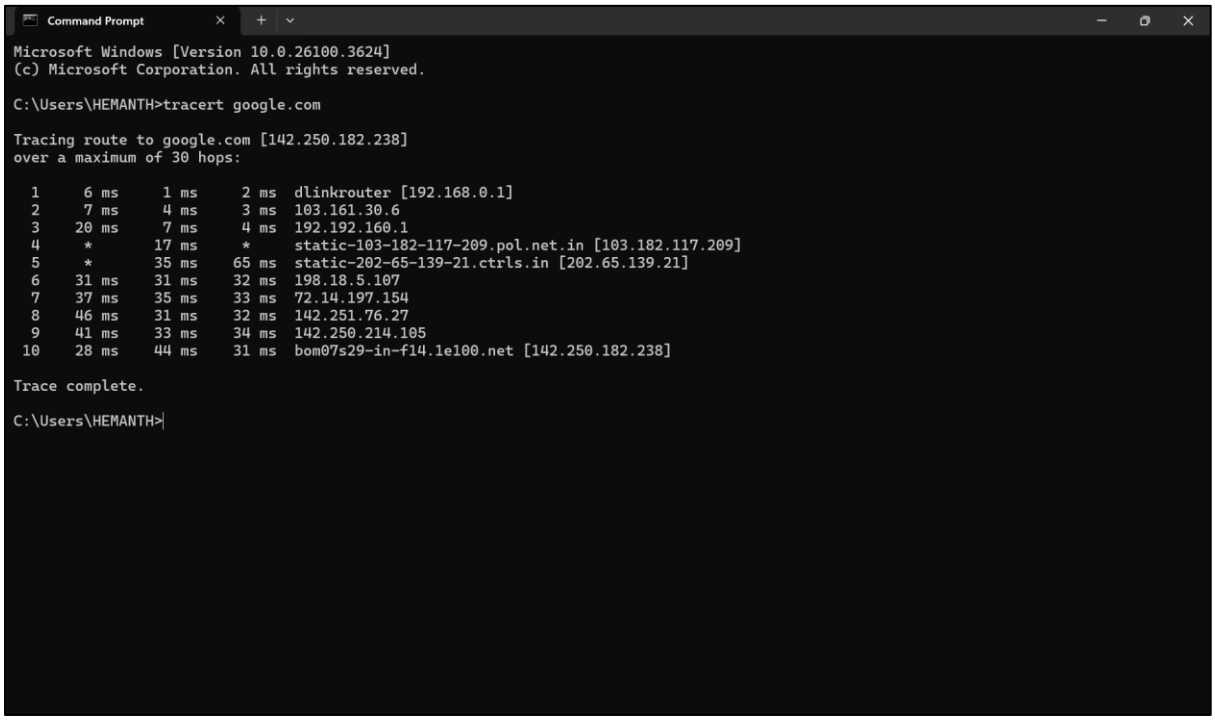


Tracent: The traceroute command (or tracert in Windows) is used to trace the route that packets take from the source to the destination across a network. It shows each intermediate hop (router or server)

the packets pass through on their way to the destination.

Ex:

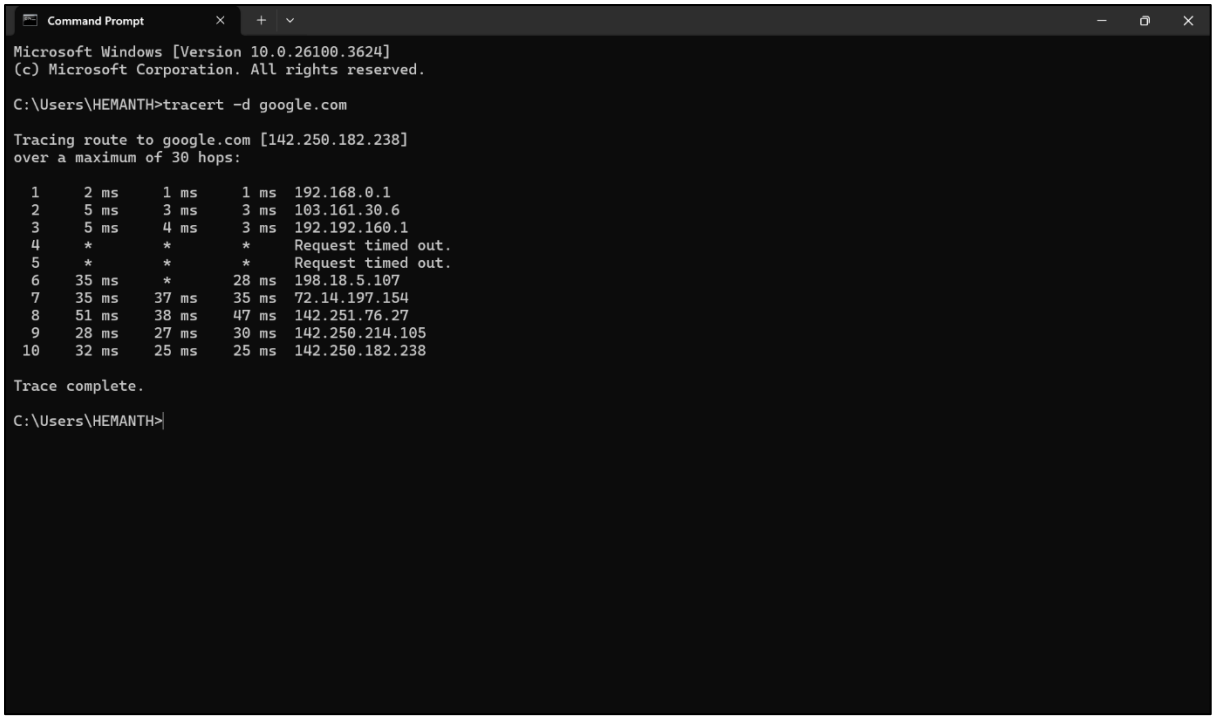
C:\Users\HEMANTH>tracert google.com



-d: The -d option in the tracert command (Windows) is used to disable reverse DNS lookups during the trace process. By default, tracert attempts to resolve IP addresses to domain names (reverse DNS lookup) for each hop. Using the -d flag prevents this, which can speed up the trace process, as it avoids waiting for DNS queries to resolve.

Ex:

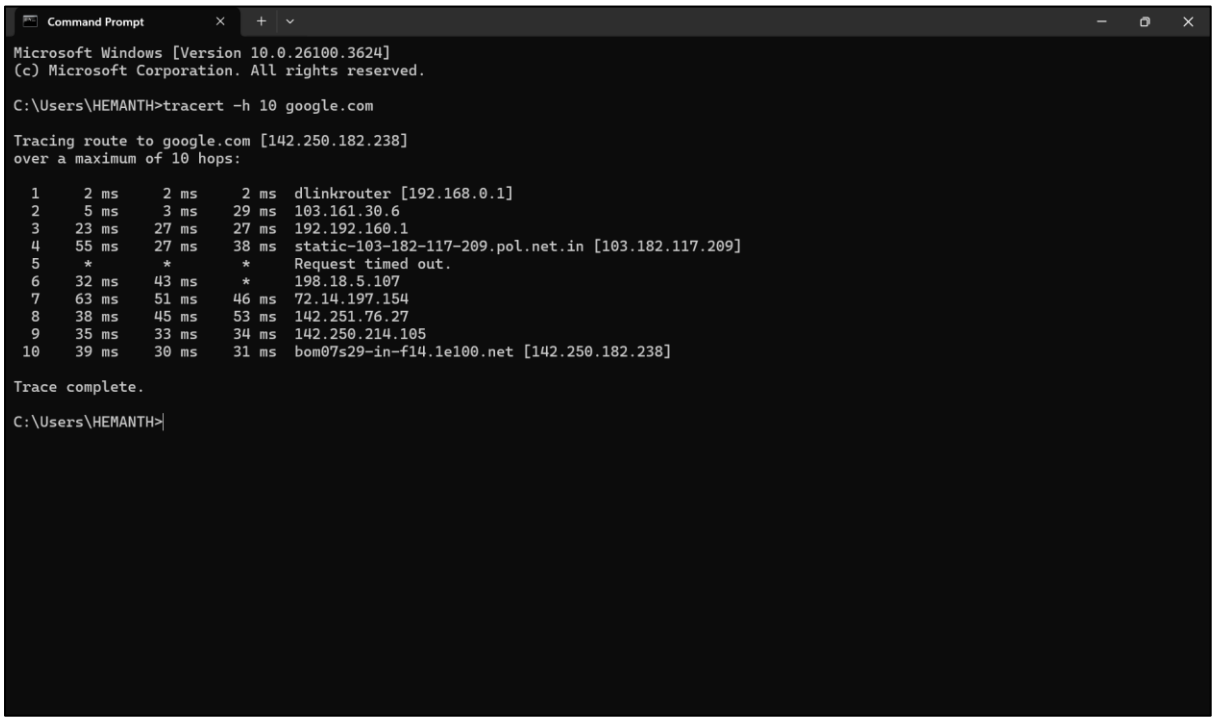
C:\Users\HEMANTH>tracert -d google.com



-h: The -h option in the tracert command (Windows) is used to specify the maximum number of hops that the traceroute should attempt before stopping. The default maximum hop count is 30, but you can change it with the -h flag.

Ex:

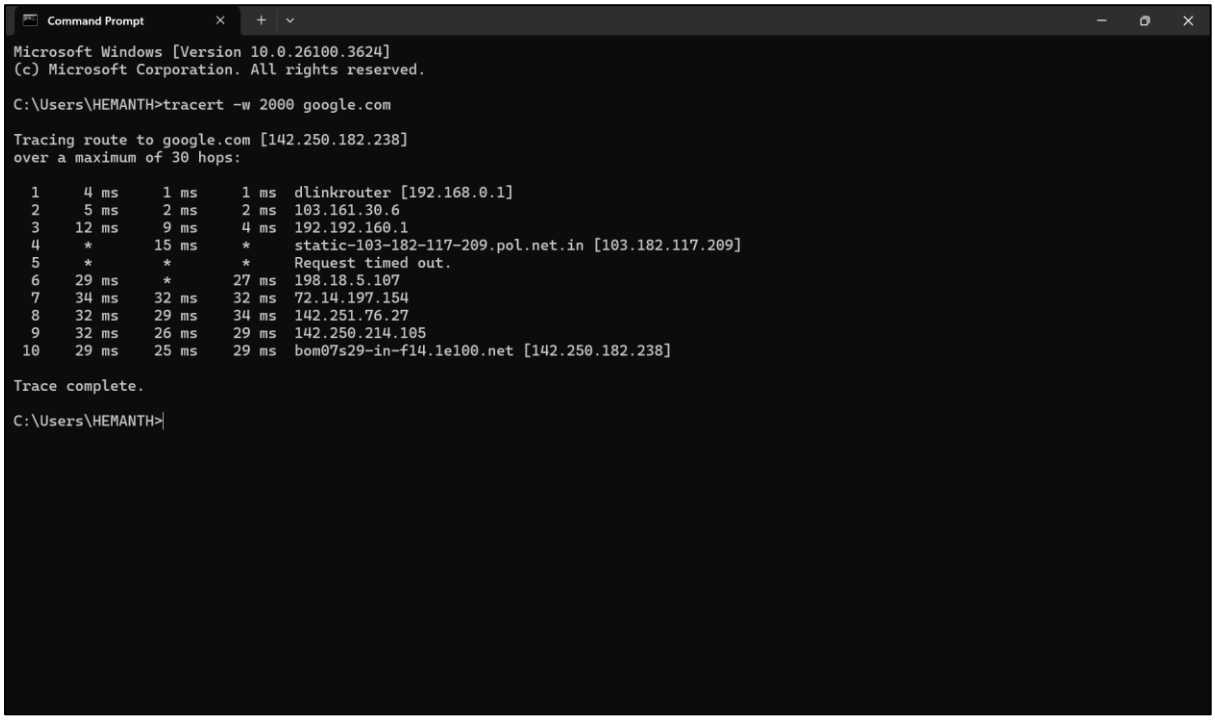
C:\Users\HEMANTH>tracert -h 10 google.com



-w: The -w option in the tracert command (Windows) is used to set the timeout in milliseconds for each reply. This determines how long tracert should wait for a response from each hop before timing out.

Ex:

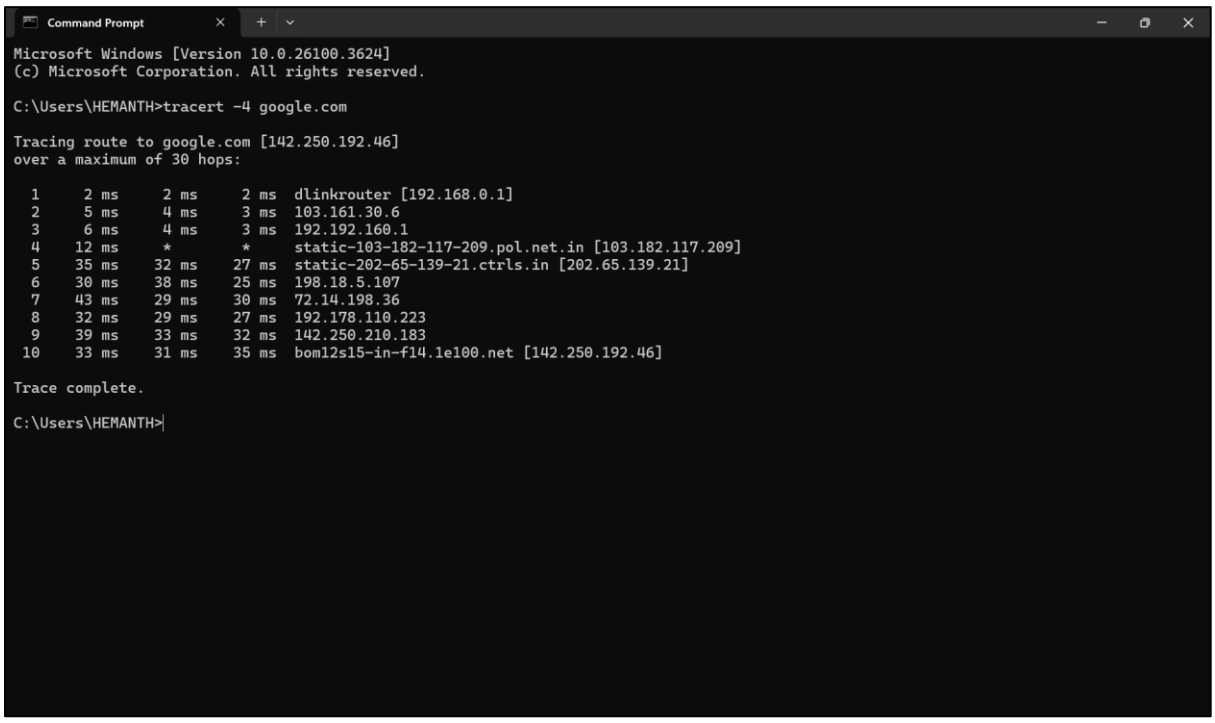
C:\Users\HEMANTH>tracert -w 2000 google.com



-4: The -4 option in the tracert command (Windows) forces the traceroute to use only IPv4 addresses, even if the destination supports both IPv4 and IPv6

Ex:

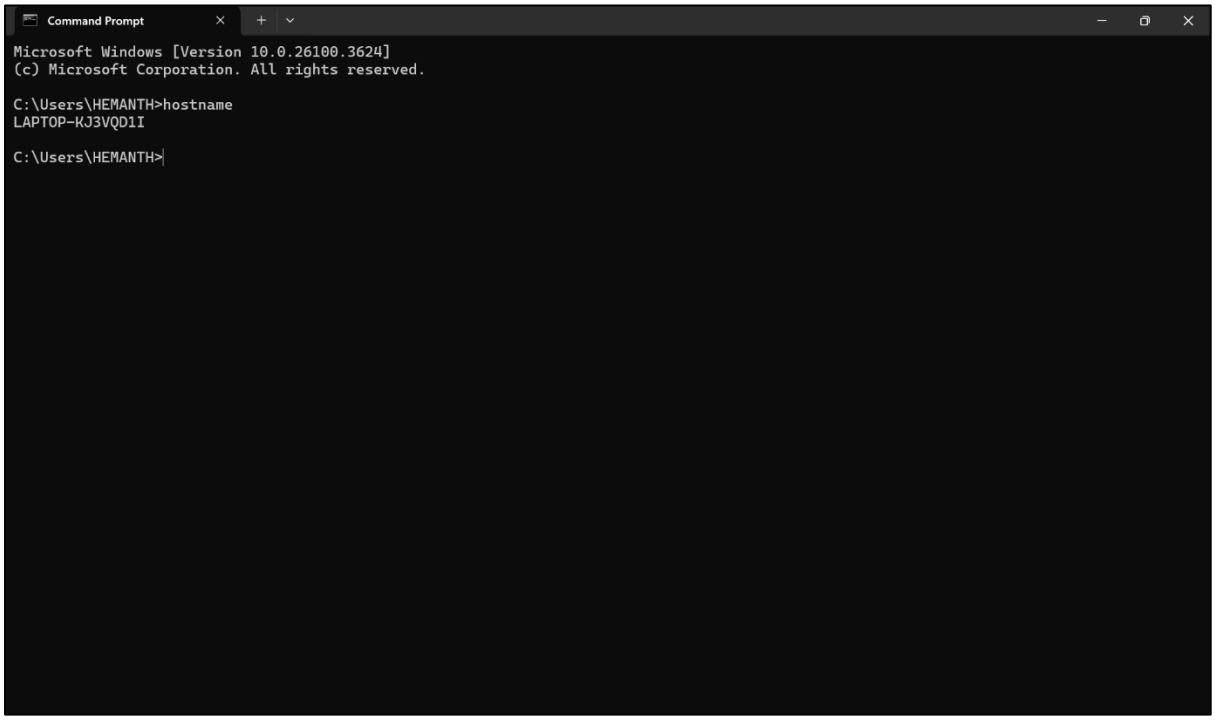
C:\Users\HEMANTH>tracert -4 google.com



Hostname: The hostname command is used to display or change the name of acomputer on a network.

Ex:

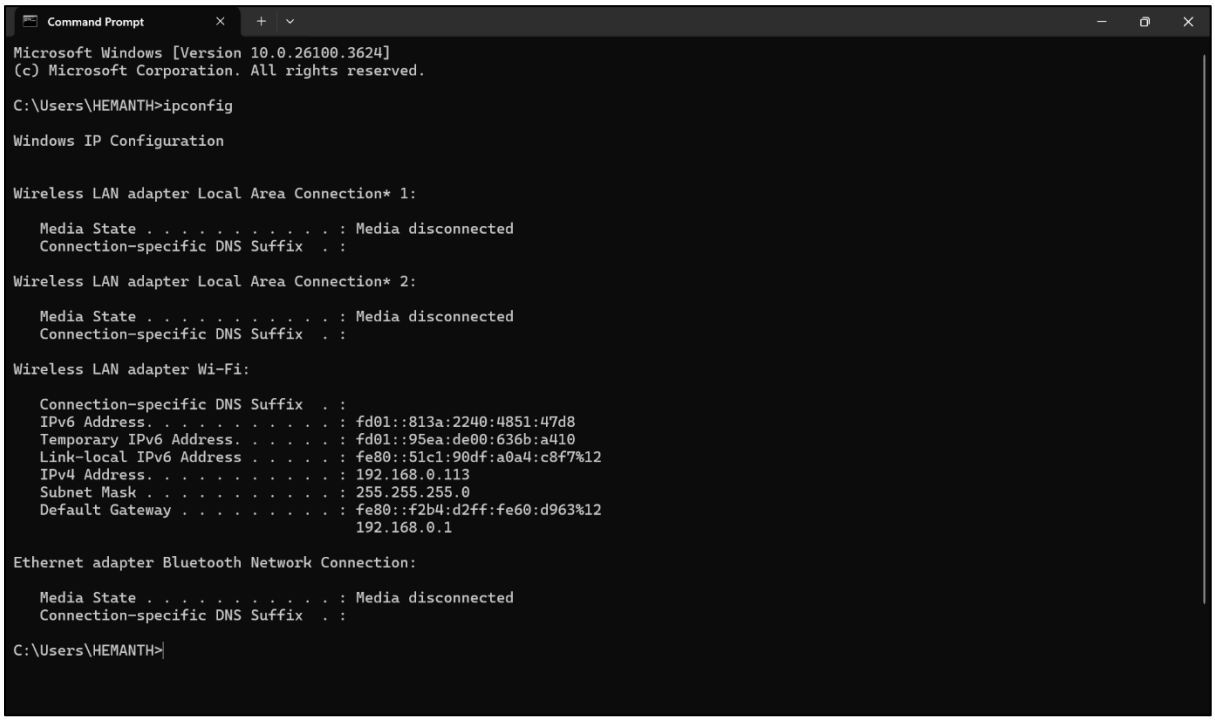
C:\Users\HEMANTH>hostname



IPConfig: The ipconfig command is a network configuration utility available in Windows. It is used to display and manage the IP address, subnet mask, default gateway, and other network settings of a device.

Ex:

C:\Users\HEMANTH>ipconfig

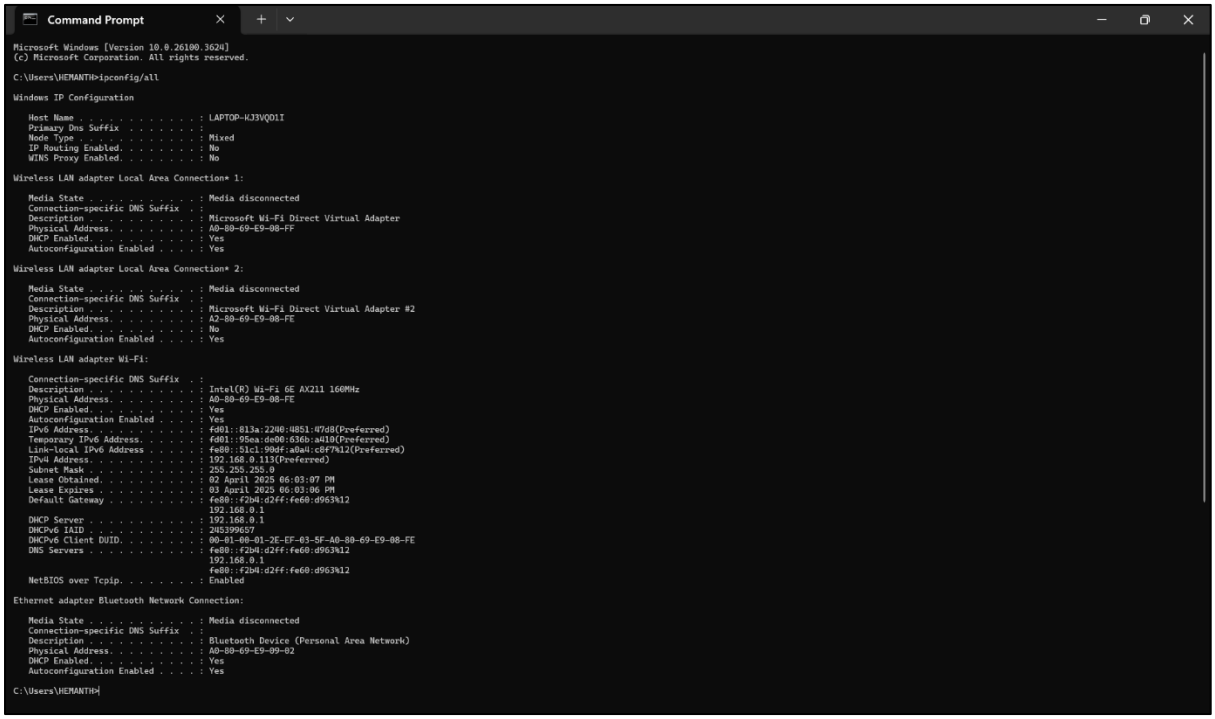


ipconfig /all : The ipconfig /all command in Windows provides a comprehensive view of the network configuration for all network interfaces on the system. It displays detailed

information, such as the IP address, subnet mask, default gateway, and other key network settings, including DNS servers and DHCP status.

Ex:

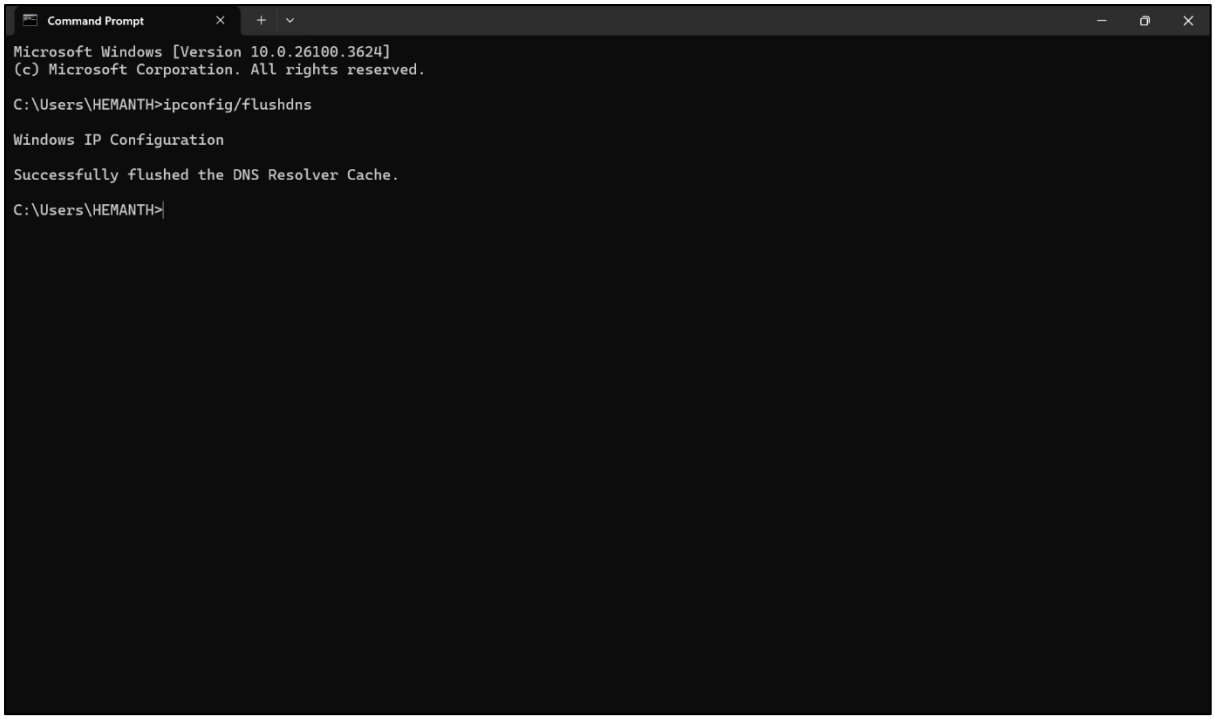
C:\Users\HEMANTH>ipconfig/all



ipconfig /flushdns: The ipconfig /flushdns command is used in Windows to clear the DNS resolver cache, which stores previously resolved domain names to IP addresses. This helps in resolving DNS-related issues by forcing the system to retrieve fresh DNS information.

Ex:

C:\Users\HEMANTH>ipconfig/flushdns

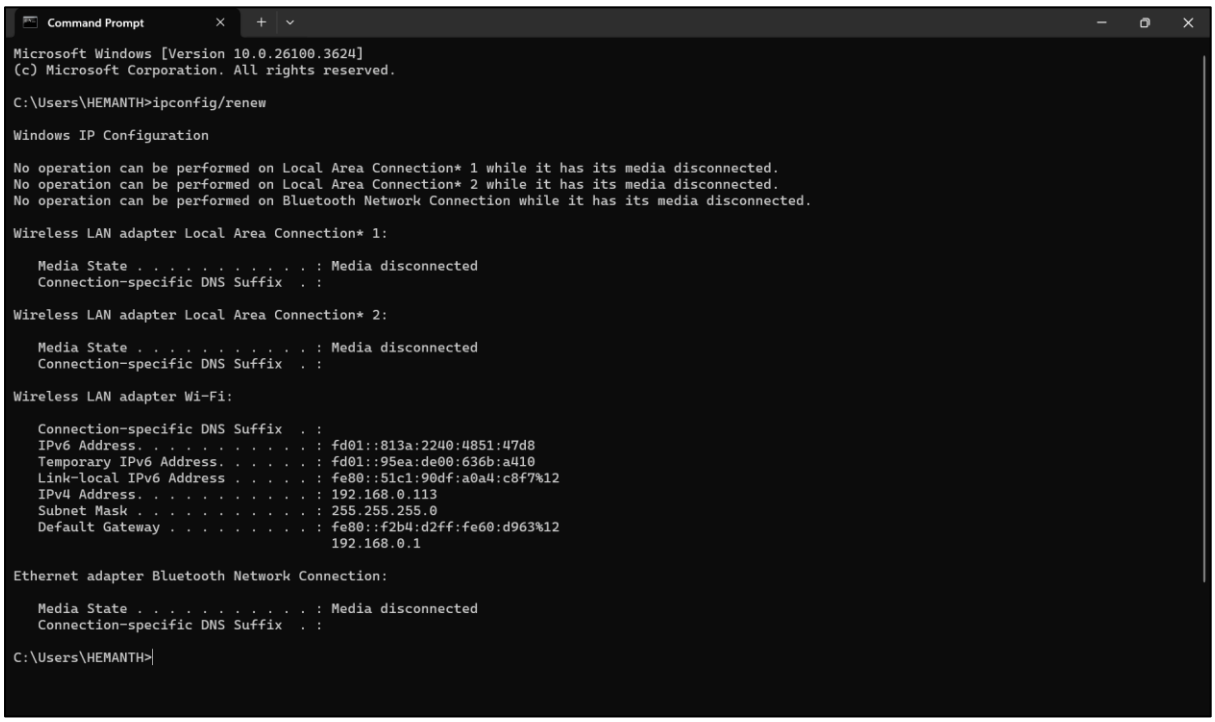


ipconfig /renew:

The ipconfig /renew command in Windows is used to request a new IP address from the DHCP server for a network adapter. This is useful when troubleshooting network issues, especially if you are experiencing connectivity problems.

Ex:

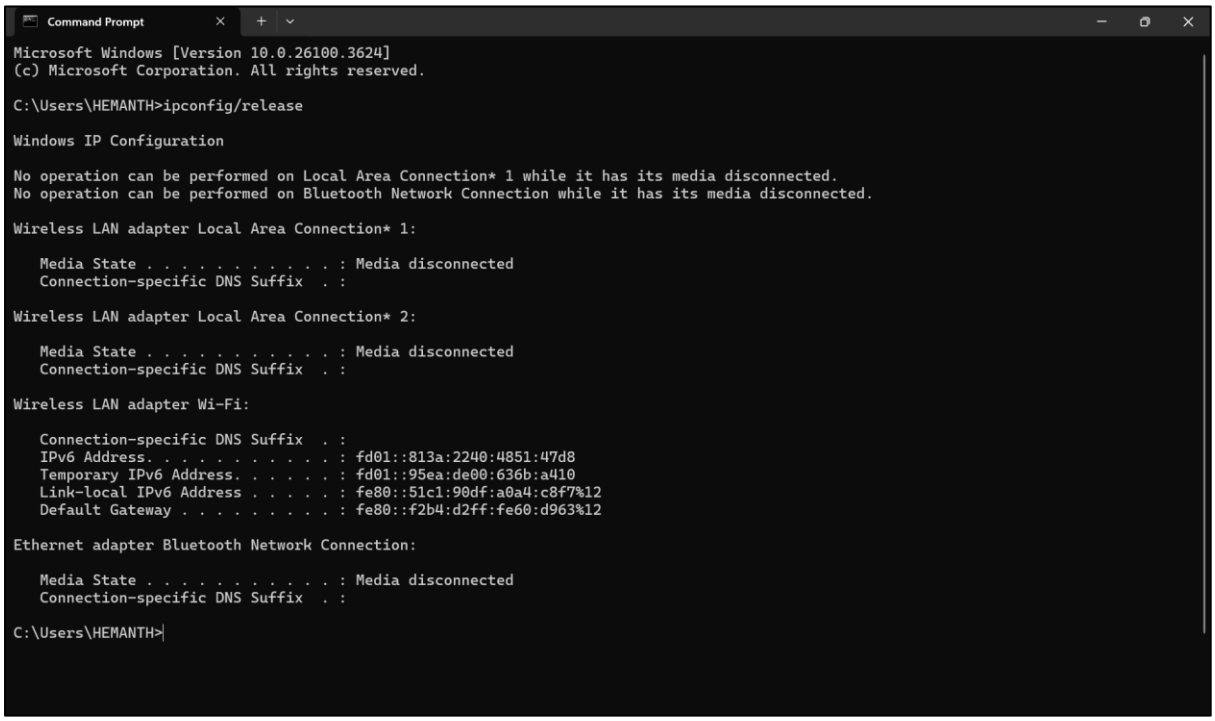
C:\Users\HEMANTH>ipconfig/renew



ipconfig /release: The ipconfig /release command is used to release the current IP address assigned by the DHCP server. This is useful when troubleshooting network issues or switching to a new network.

Ex:

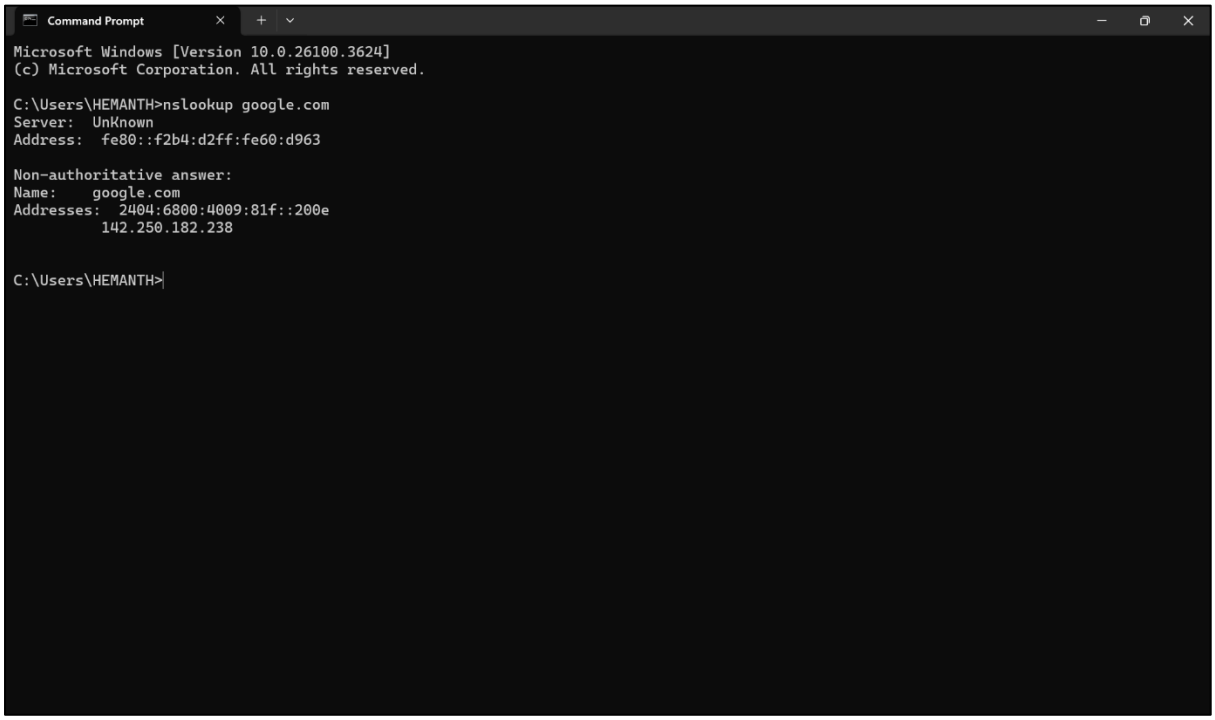
C:\Users\HEMANTH>ipconfig/release



Nslookup: The nslookup (Name Server Lookup) command is used to query the Domain Name System (DNS) to obtain domain name or IP address mapping details. It is useful for troubleshooting DNS-related issues and checking domain records.

Ex:

C:\Users\HEMANTH>nslookup google.com

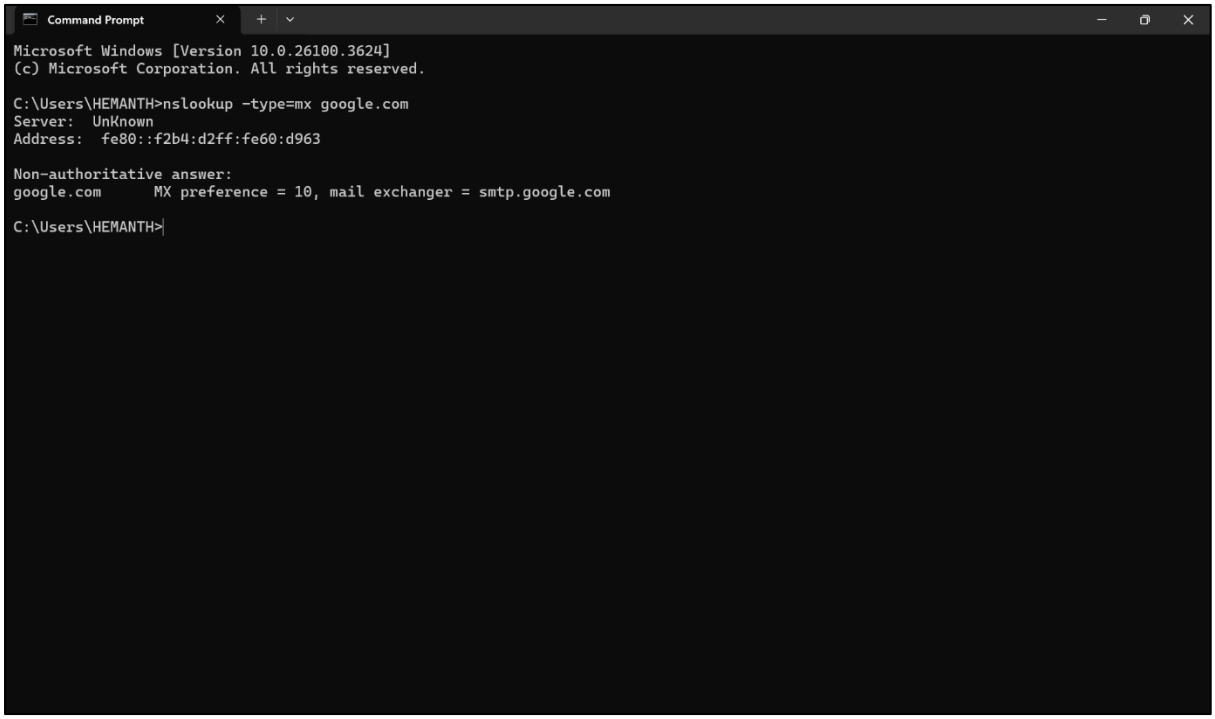


-type=<record>: In nslookup, the -type=<record> option is used to specify the type of DNS record you want to query. This allows you to retrieve different types of DNS

information, such as A (IPv4), AAAA (IPv6), MX (Mail Exchange), TXT (Text), and more.

Ex:

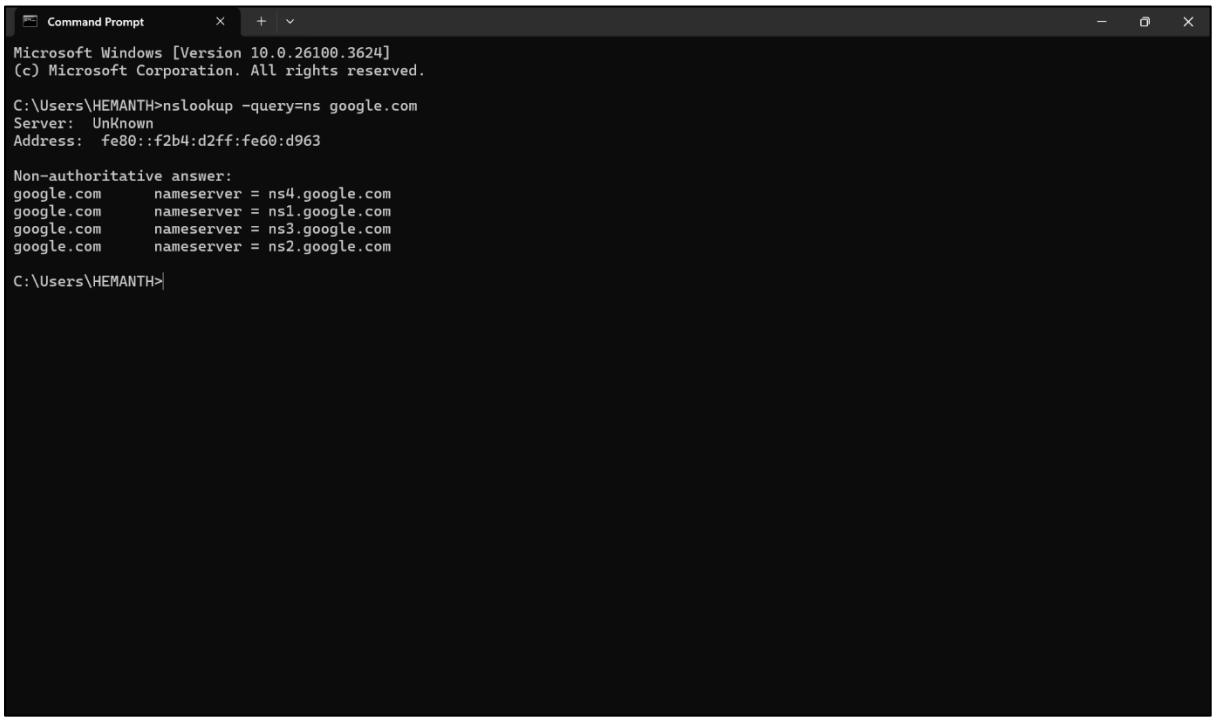
C:\Users\HEMANTH>nslookup -type=mx google.com



-query=<record>: The -query=<record> option in nslookup is an alternative way to specify the type of DNS record you want to retrieve. It works the same way as -type=<record> and allows you to query different DNS record types.

Ex:

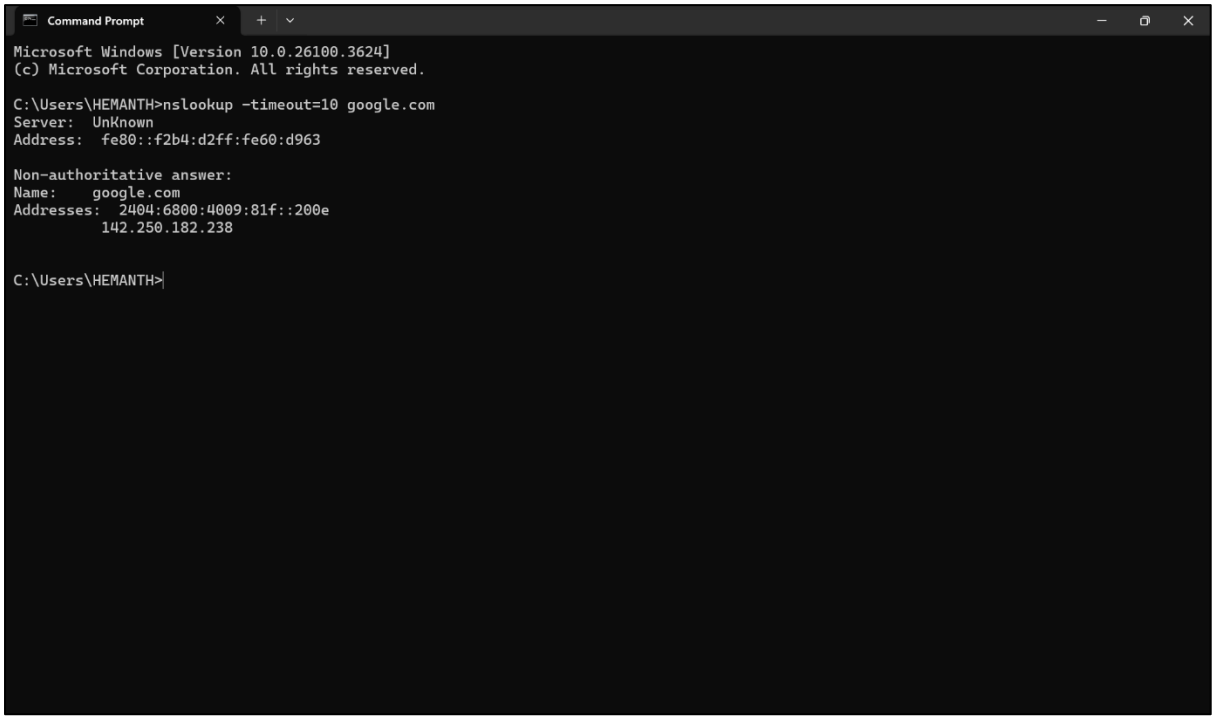
C:\Users\HEMANTH>nslookup -query=ns google.com



-timeout<seconds>: The -timeout=<seconds> option in nslookup is used to specify the number of seconds the command should wait for a response from a DNS server before timing out. This is useful when troubleshooting slow or unresponsive DNS queries.

Ex:

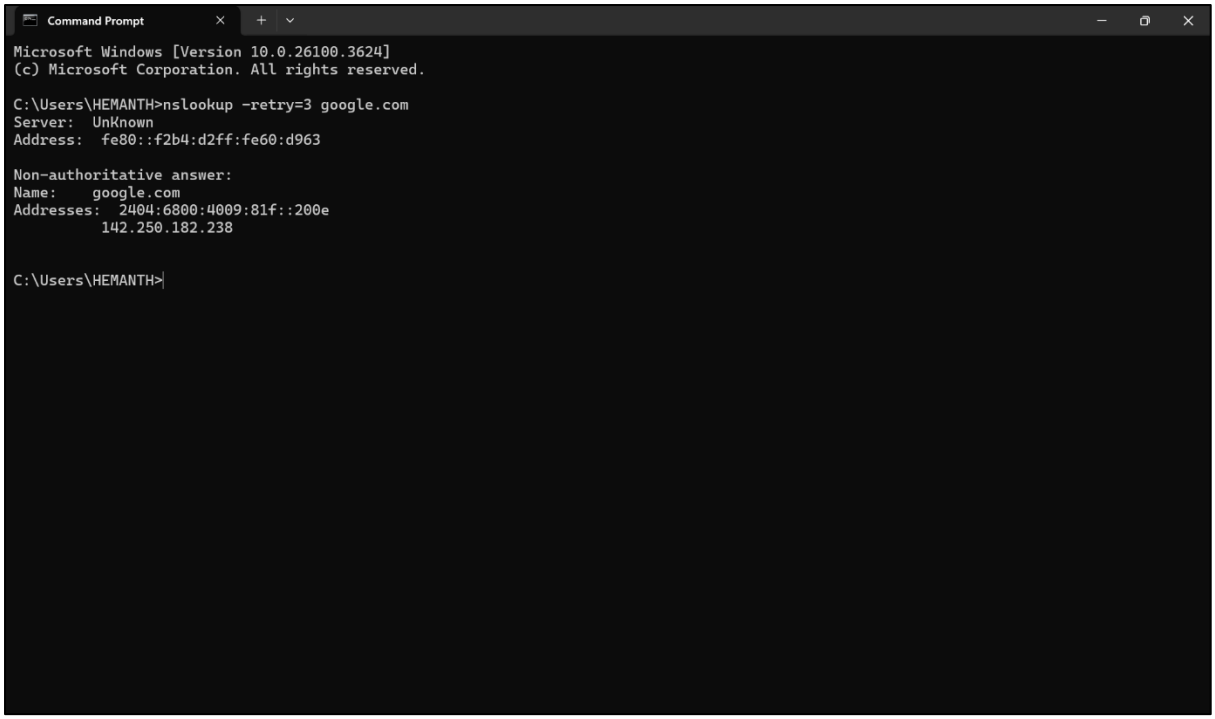
C:\Users\HEMANTH>nslookup -timeout=10 google.com



-retry=<count>: The -retry=<count> option in nslookup specifies the number of times the command will attempt to contact the DNS server before giving up if it does not receive a response. This is useful for handling intermittent network issues or slow DNS responses.

Ex:

C:\Users\HEMANTH>nslookup -retry=3 google.com

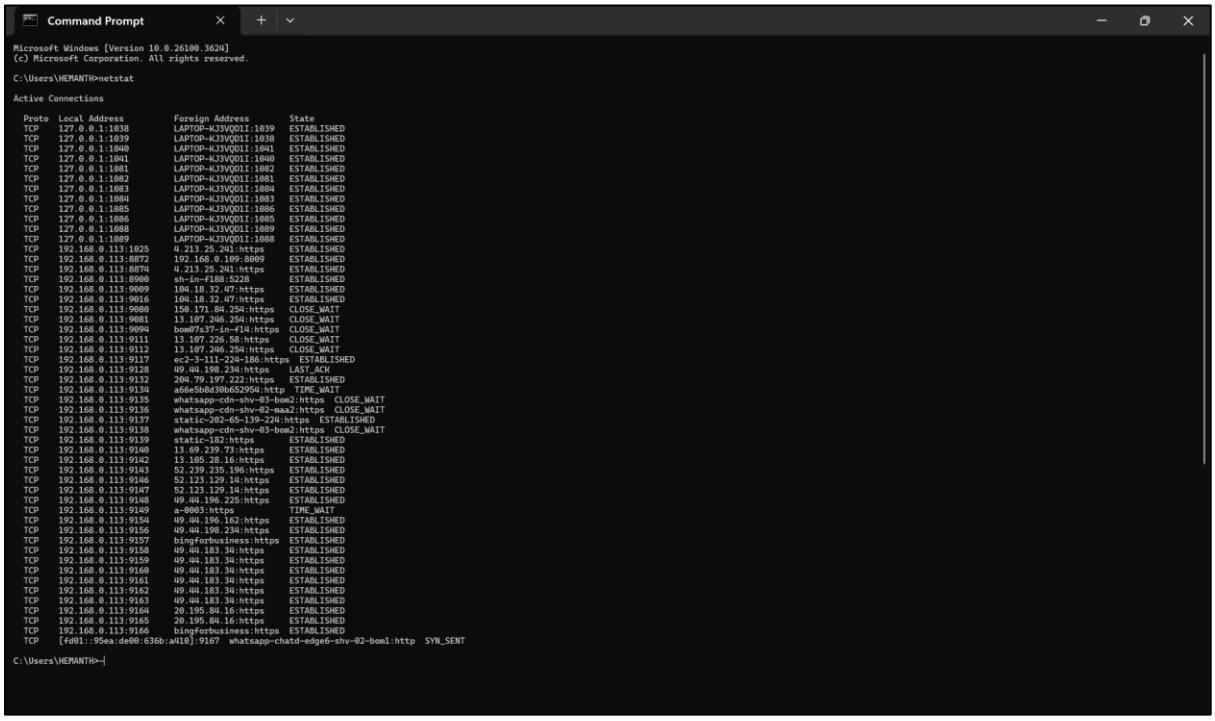


netstat :

netstat is a command-line tool used to display network connections, routing tables, interface statistics, and network protocol information. It helps monitor and troubleshoot network-related issues on a system.

Ex:

C:\Users\HEMANTH>netstat

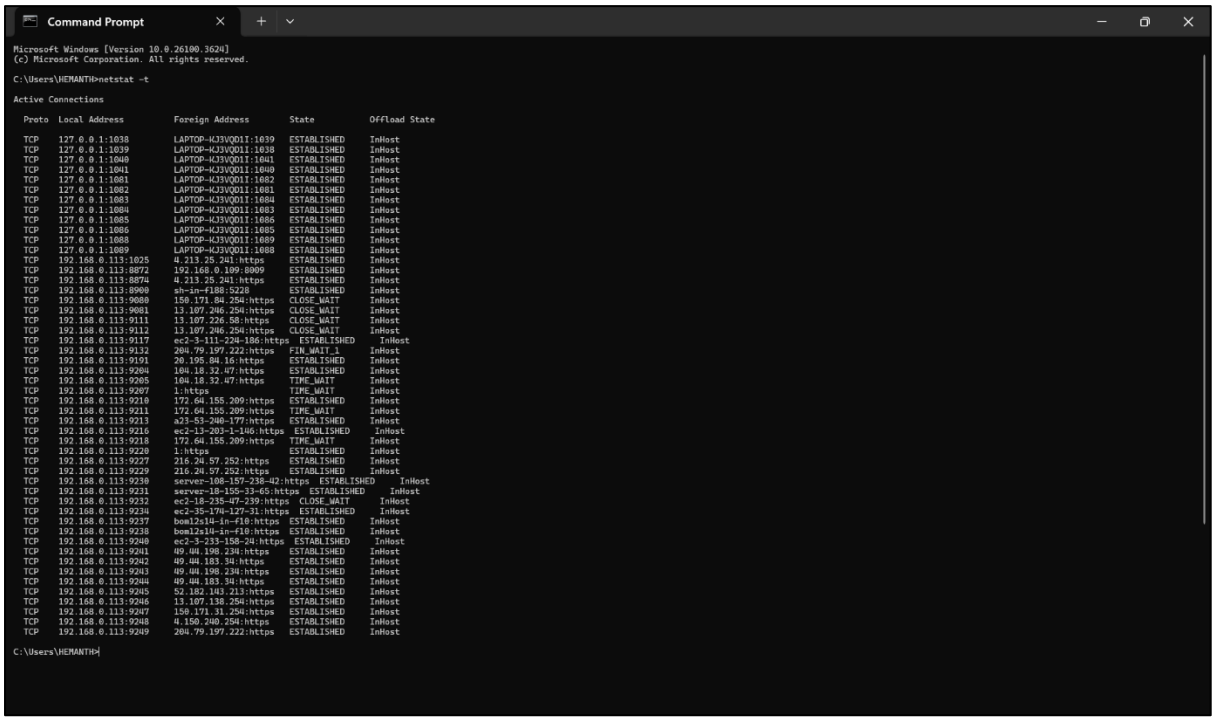


-t: The **-t** option in the **netstat** command is used to display TCP (Transmission Control Protocol) connections. When you run **netstat -t**, it will show you a list of all active TCP

connections on your system, along with their current states.

Ex:

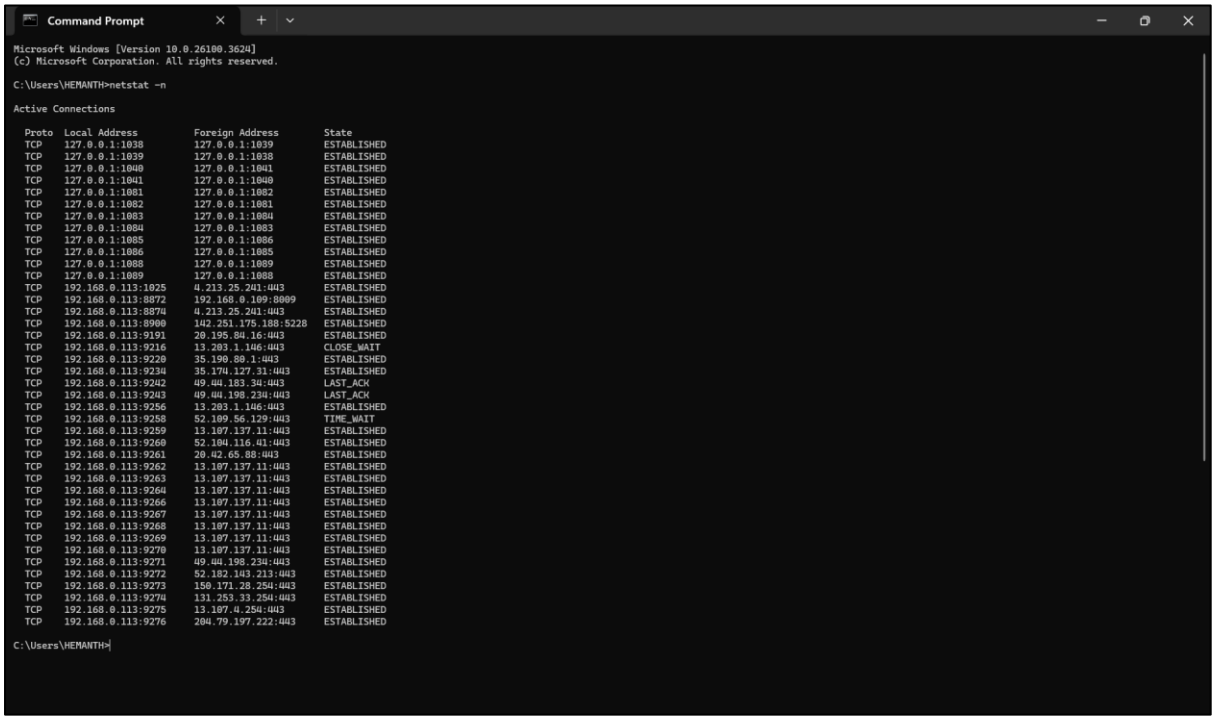
C:\Users\HEMANTH>netstat -t



-n: The -n option in netstat displays network connections using numerical IP addresses and port numbers instead of resolving them to hostnames and service names.

Ex:

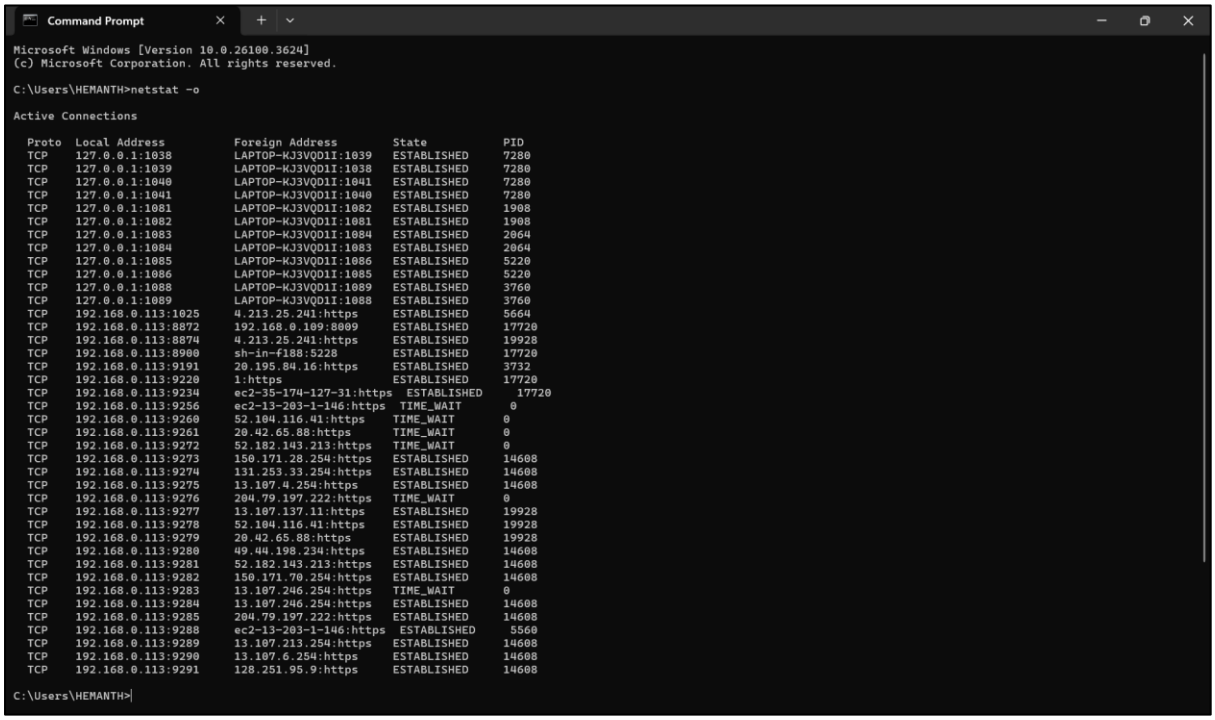
C:\Users\HEMANTH>netstat -n



-o: netstat -o shows active network connections along with the Process ID (PID), helping identify which process is using a specific port.

Ex:

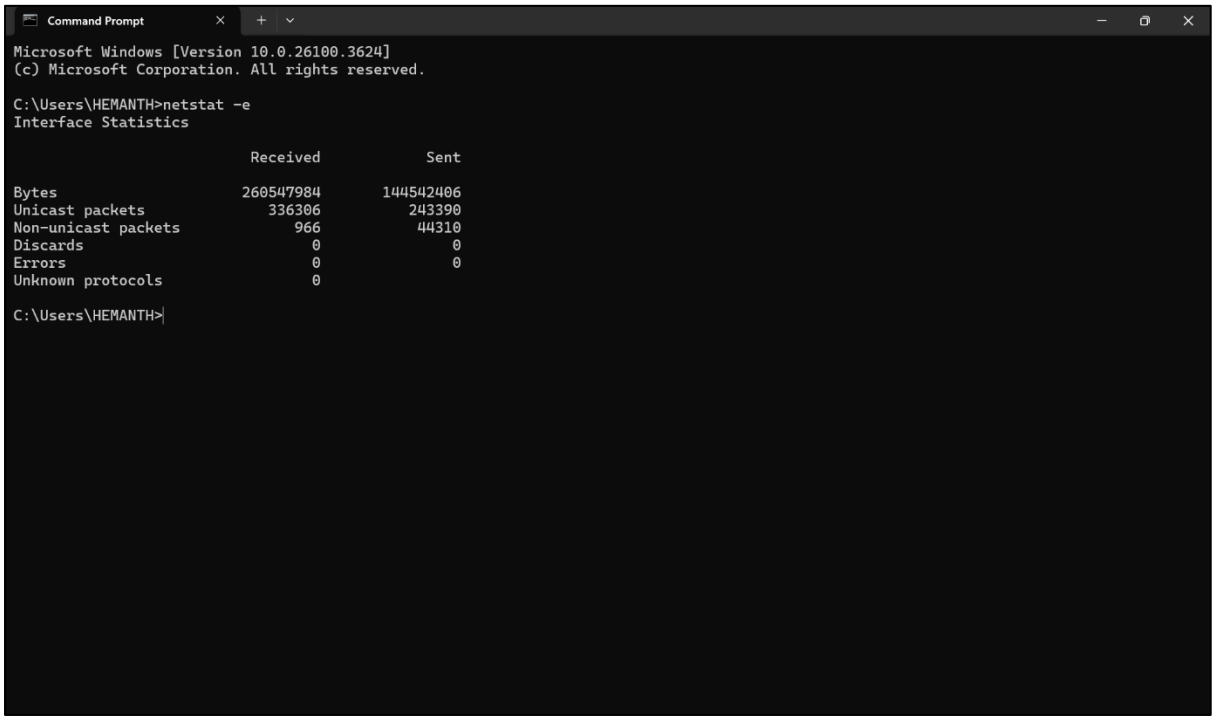
C:\Users\HEMANTH>netstat -o



-e: The -e option in the netstat command is used to display network interface statistics, including the number of packets sent and received, errors, and dropped packets.

Ex:

C:\Users\HEMANTH>netstat -e



Experiment-2

AIM: To implement CRC for error delection using the CRC-CCIT generator.

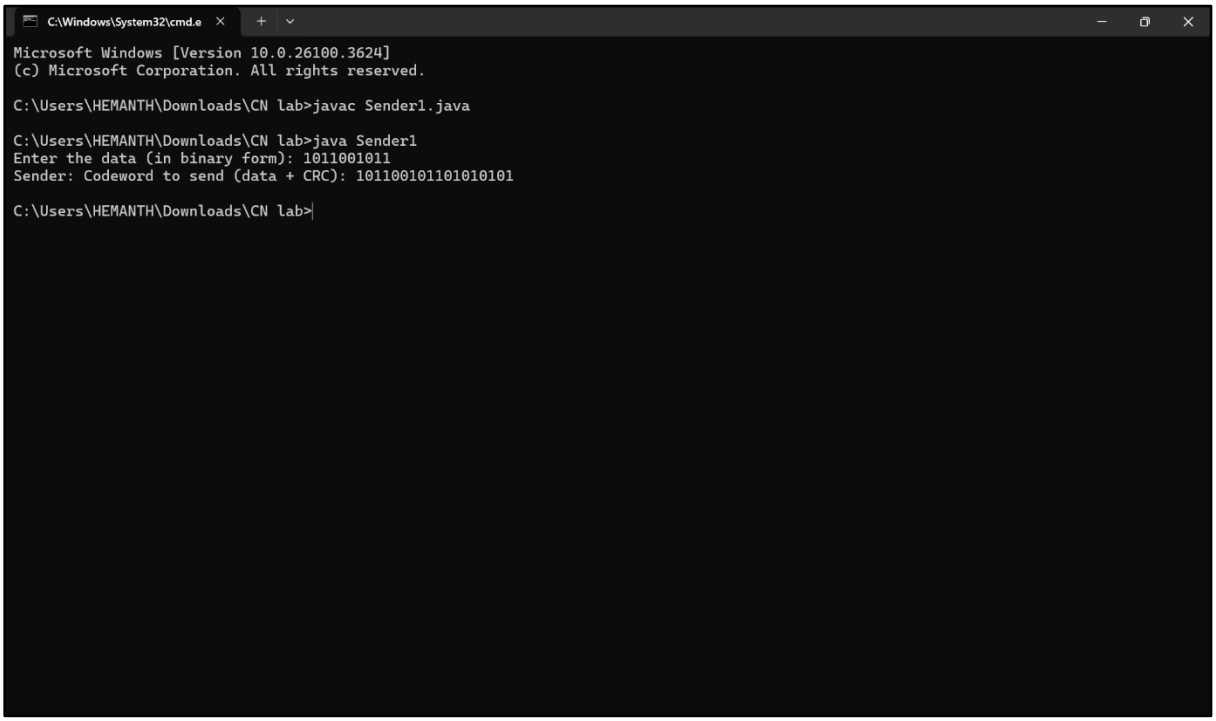
Description: Cyclic Redundancy Check (CRC) is a method used to detect errors in digital data. The **CRC-CCITT** generator is a standard polynomial used in many communication systems.

Sender:

```
import java.util.Scanner;
public class Sender1 {
   static final String POLYNOMIAL = "100000111";
   static final int POLY_LENGTH = POLYNOMIAL.length();
   static String xor(String a, String b) {
      StringBuilder result = new StringBuilder();
      for (int i = 0; i < a.length(); i++) {
         result.append(a.charAt(i) == b.charAt(i) ? '0' : '1');
      }
      return result.toString();
   }
   static String calculateCRC(String data) {
      StringBuilder dataWithZeros = new StringBuilder(data);
      for (int i = 0; i < POLY_LENGTH - 1; i++) {
         dataWithZeros.append('0');
      }
      String remainder = dataWithZeros.toString();
      for (int i = 0; i < data.length(); i++) {
         if (remainder.charAt(i) == '1') {
            remainder = remainder.substring(0, i) + xor(remainder.substring(i, i + POLY_LENGTH),
POLYNOMIAL) + remainder.substring(i + POLY_LENGTH);
         }
      }
      return remainder.substring(data.length(), remainder.length());
   }
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter the data (in binary form): ");
      String data = scanner.nextLine();
      if (!data.matches("[01]+")) {
         System.out.println("Invalid input. Please enter a binary string.");
```

```
            return;
        }
        String crc = calculateCRC(data);
        String codeword = data + crc;
        System.out.println("Sender: Codeword to send (data + CRC): " + codeword);
        scanner.close();
    }
}
```

Output:

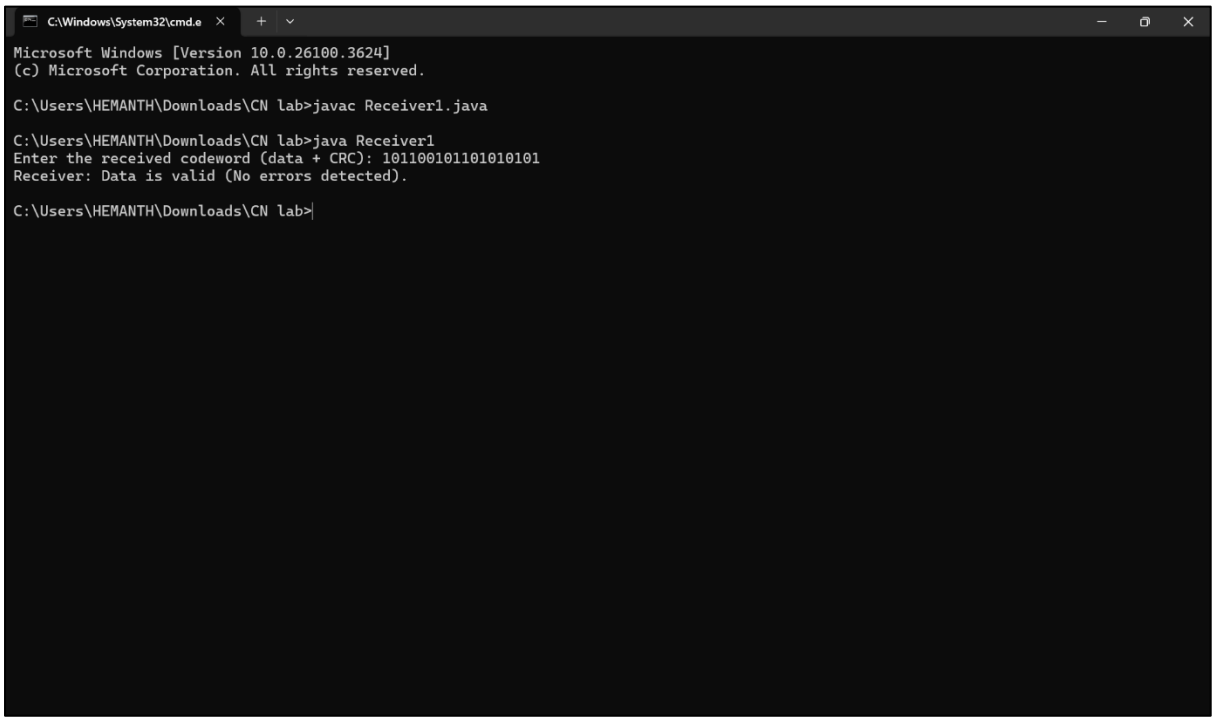


Receiver:

```
import java.util.Scanner;
public class Receiver1 {
    static final String POLYNOMIAL = "100000111";
    static final int POLY_LENGTH = POLYNOMIAL.length();
    static String xor(String a, String b) {
        StringBuilder result = new StringBuilder();
        for (int i = 0; i < a.length(); i++) {
            result.append(a.charAt(i) == b.charAt(i) ? '0' : '1');
        }
        return result.toString();
    }
    static String calculateCRC(String data) {
        StringBuilder dataWithZeros = new StringBuilder(data);
        for (int i = 0; i < POLY_LENGTH - 1; i++) {
            dataWithZeros.append('0');
        }
        String remainder = dataWithZeros.toString();
        for (int i = 0; i < data.length(); i++) {
            if (remainder.charAt(i) == '1') {
                remainder = remainder.substring(0, i) + xor(remainder.substring(i, i + POLY_LENGTH),
POLYNOMIAL) + remainder.substring(i + POLY_LENGTH);
            }
        }
        return remainder.substring(data.length(), remainder.length());
    }
    static boolean checkCRC(String dataWithCRC) {
        String crc = calculateCRC(dataWithCRC);
        return crc.equals("00000000");
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the received codeword (data + CRC): ");
        String receivedData = scanner.nextLine();
        if (!receivedData.matches("[01]+")) {
            System.out.println("Invalid input. Please enter a binary string.");
            return;
```

```
        }
        if (checkCRC(receivedData)) {
            System.out.println("Receiver: Data is valid (No errors detected).");
        } else {
            System.out.println("Receiver: Data is invalid (Errors detected).");
        }
        scanner.close();
    }
}
```

Output:



Experiment-3

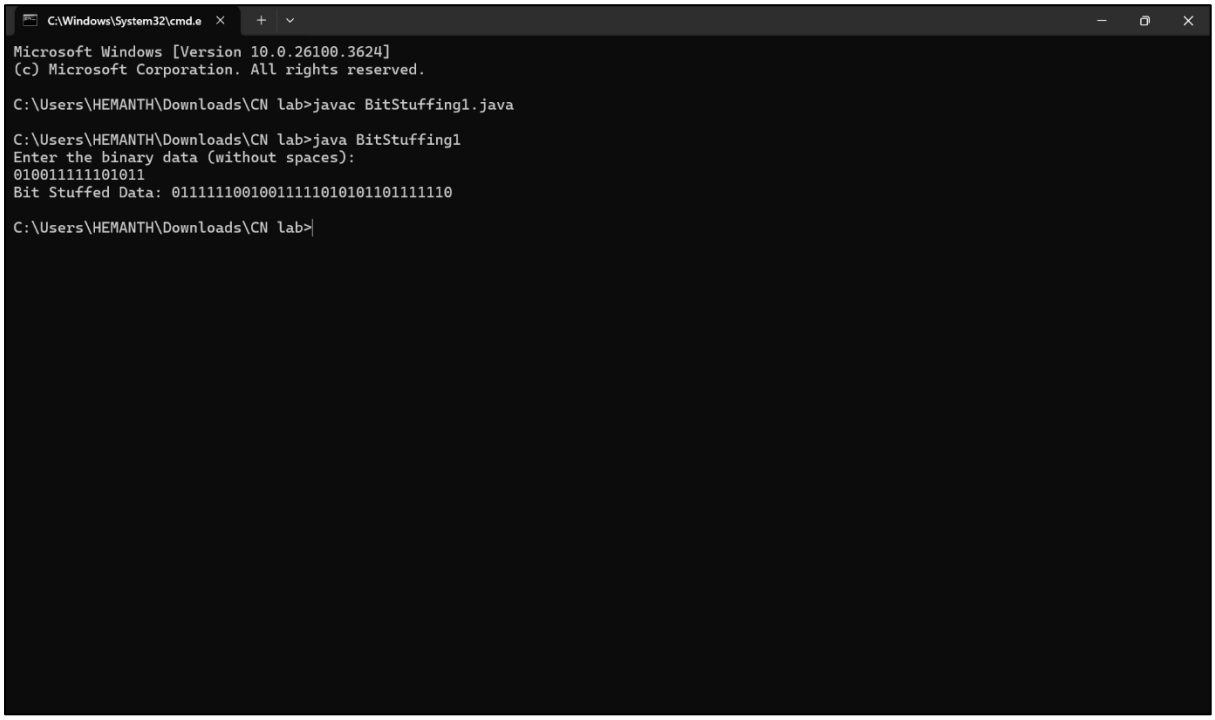
AIM: To implement data-link layer sharing for bit-stuffing.

Description: Bit stuffing is a technique used in data communication to prevent confusion with special control sequences. When sending data, a '0' is inserted after five consecutive '1's in the data stream. This ensures that special patterns (like frame delimiters) are not accidentally interpreted as part of the data.At the receiver's end, the stuffed '0's are removed to recover the original data.

BitStuffing:

```
import java.util.Scanner;
public class BitStuffing1 {
    public static String bitStuffing(String inputData) {
        StringBuilder stuffedData = new StringBuilder();
        int count = 0;
        stuffedData.append("01111110");
        for (int i = 0; i < inputData.length(); i++) {
            stuffedData.append(inputData.charAt(i));
            if (inputData.charAt(i) == '1') {
                count++;
                if (count == 5) {
                    stuffedData.append('0');
                    count = 0;
                }
            } else {
                count = 0;
            }
        }
        stuffedData.append("01111110");
        return stuffedData.toString();
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the binary data (without spaces):");
        String inputData = scanner.nextLine();
        String stuffedData = bitStuffing(inputData);
        System.out.println("Bit Stuffed Data: " + stuffedData);
    }
}
```

Output:

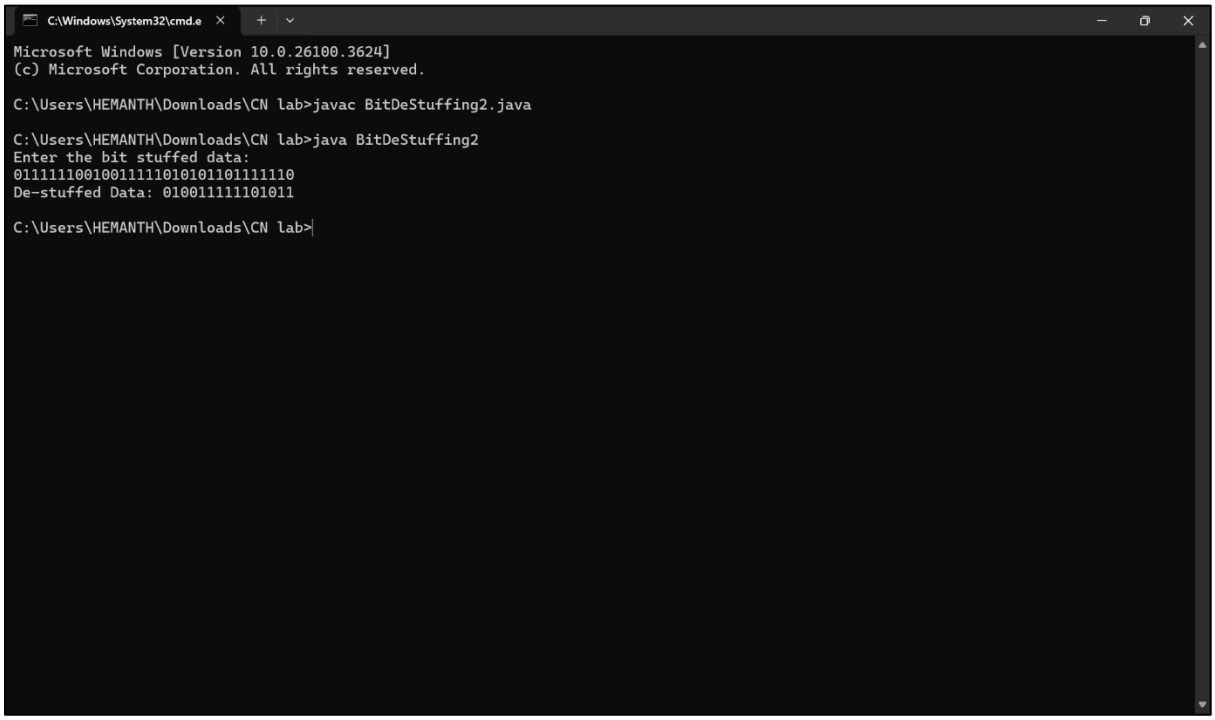

C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.26100.3624]
(c) Microsoft Corporation. All rights reserved.
C:\Users\HEMANTH\Downloads\CN lab>javac BitStuffing1.java
C:\Users\HEMANTH\Downloads\CN lab>java BitStuffing1
Enter the binary data (without spaces):
010011111101011
Bit Stuffed Data: 011111100100111110101011011111110
C:\Users\HEMANTH\Downloads\CN lab>

Description: Bit destuffing is the process used by the receiver to remove the extra bits added during bit stuffing. After detecting five consecutive '1's in the received data, the receiver removes the '0' that follows. This restores the original data before it was transmitted.

BitDestuffing:

```
import java.util.Scanner;
public class BitDeStuffing2 {
    public static String bitDeStuffing(String stuffedData) {
        StringBuilder deStuffedData = new StringBuilder();
        int count = 0;
        String dataWithoutFlags = stuffedData.substring(8, stuffedData.length() - 8);
        for (int i = 0; i < dataWithoutFlags.length(); i++) {
            deStuffedData.append(dataWithoutFlags.charAt(i));
            if (dataWithoutFlags.charAt(i) == '1') {
                count++;
                if (count == 5 && i + 1 < dataWithoutFlags.length() && dataWithoutFlags.charAt(i + 1) ==
'0') {
                    i++;
                    count = 0;
                }
            } else {
                count = 0;
            }
        }
        return deStuffedData.toString();
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the bit stuffed data:");
        String stuffedData = scanner.nextLine();
        String deStuffedData = bitDeStuffing(stuffedData);
        System.out.println("De-stuffed Data: " + deStuffedData);
    }
}
```

Output:


C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.26100.3624]
(c) Microsoft Corporation. All rights reserved.
C:\Users\HEMANTH\Downloads\CN lab>javac BitDeStuffing2.java
C:\Users\HEMANTH\Downloads\CN lab>java BitDeStuffing2
Enter the bit stuffed data:
01111110010011111010101101111110
De-stuffed Data: 010011111101011
C:\Users\HEMANTH\Downloads\CN lab>

Experiment-4

AIM: To implement byte stuffing (or) character stuffing.

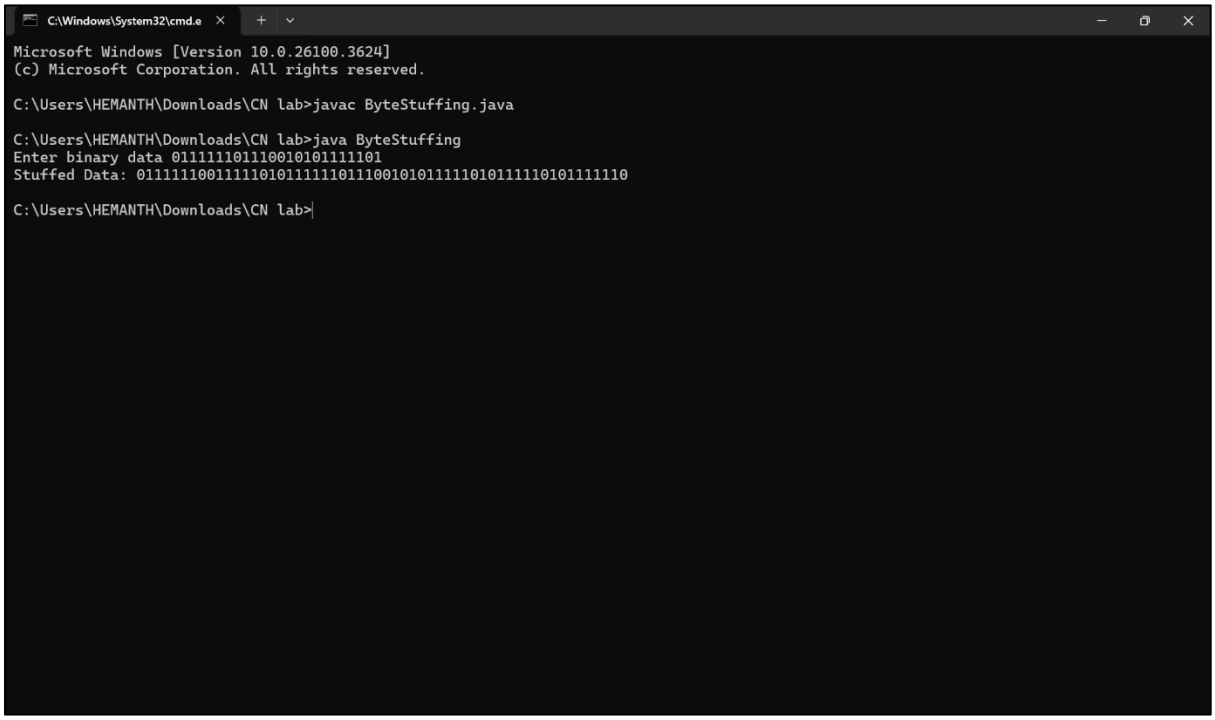
Description: Byte stuffing is a technique used to avoid confusion with special control bytes (like frame start or end). When a special byte appears in the data, an **escape byte** is added before it. The receiver removes the escape byte during decoding to get the original data.

ByteStuffing:

```
import java.util.Scanner;
public class ByteStuffing {
    private static final String FLAG = "01111110";
    private static final String ESCAPE = "01111101";
    public static String bitStuffing(String data) {
        StringBuilder stuffedData = new StringBuilder(FLAG);
        for (int i = 0; i < data.length(); i += 8) {
            String byteChunk = data.substring(i, Math.min(i + 8, data.length()));
            if (byteChunk.equals(FLAG) || byteChunk.equals(ESCAPE)) {
                stuffedData.append(ESCAPE);
            }
            stuffedData.append(byteChunk);
        }
        stuffedData.append(FLAG);
        return stuffedData.toString();
    }
    public static boolean isValidBinary(String data) {
        return data.matches("[01]+");
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter binary data ");
        String input = scanner.nextLine();
        if (!isValidBinary(input)) {
            System.out.println("Invalid input! Please enter only binary digits (0's and 1's).");
        } else {
            String stuffedData = bitStuffing(input);
            System.out.println("Stuffed Data: " + stuffedData);
        }
        scanner.close();
```

```
    }
}
```

Output:


C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.26100.3624]
(c) Microsoft Corporation. All rights reserved.
C:\Users\HEMANTH\Downloads\CN lab>javac ByteStuffing.java
C:\Users\HEMANTH\Downloads\CN lab>java ByteStuffing
Enter binary data 011111101110010101111101
Stuffed Data: 0111111001111101011111101110010101111101011111010111110
C:\Users\HEMANTH\Downloads\CN lab>

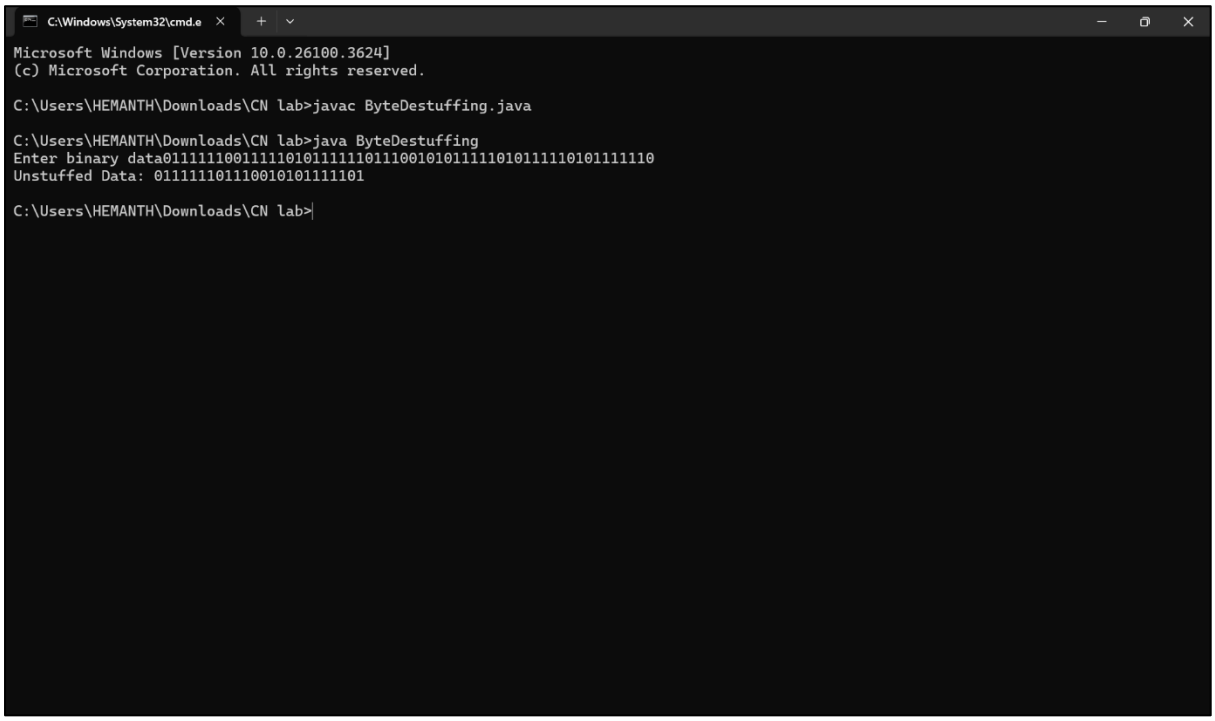
Description: Byte destuffing is the process used by the receiver to remove the escape bytes added during byte stuffing. When an escape byte is detected, the next byte is treated as data—even if it's a control character—restoring the original message.

ByteDeStuffing:

```
import java.util.Scanner;
public class ByteDestuffing {
   private static final String FLAG = "01111110";
   private static final String ESCAPE = "01111101";
   public static String bitUnstuffing(String stuffedData) {
      String data = stuffedData.substring(FLAG.length(), stuffedData.length() - FLAG.length());
      StringBuilder unstuffedData = new StringBuilder();
      int i = 0;
      while (i < data.length()) {
         if (i + ESCAPE.length() <= data.length() && data.substring(i, i +
ESCAPE.length()).equals(ESCAPE)) {
            i += ESCAPE.length();
            if (i + 8 <= data.length()) {
               unstuffedData.append(data.substring(i, i + 8));
               i += 8;
            }
         } else {
            unstuffedData.append(data.substring(i, i + 8));
            i += 8;
         }
      }
      return unstuffedData.toString();
   }
   public static boolean isValidBinary(String data) {
      return data.matches("[01]+");
   }
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter binary data");
      String input = scanner.nextLine();
      if (!isValidBinary(input)) {
         System.out.println("Invalid input! Please enter only binary digits (0's and 1's).");
      } else {
```

```
            String unstuffedData = bitUnstuffing(input);
            System.out.println("Unstuffed Data: " + unstuffedData);
        }
        scanner.close();
    }
}
```

Output:



Experiment-5

Aim: To implement stop and wait protocol program.

Description: The Stop and Wait protocol is a simple data link layer method for reliable communication. The sender sends one frame at a time and waits for an acknowledgment (ACK) before sending the next frame. If no ACK is received, the sender retransmits the frame. This ensures error-free and orderly delivery, but is slower due to waiting after each frame.

Client:

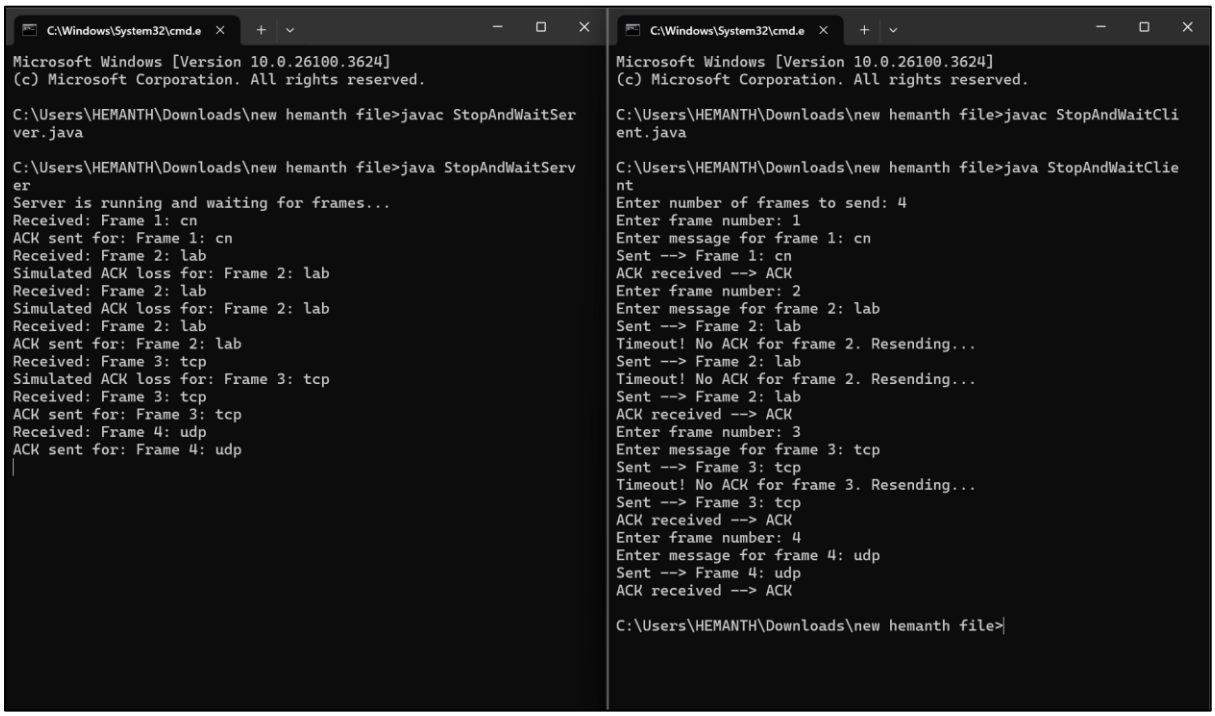
```
import java.net.*;
import java.util.*;
public class StopAndWaitClient {
    public static void main(String[] args) throws Exception {
        DatagramSocket clientSocket = new DatagramSocket();
        clientSocket.setSoTimeout(2000);
        InetAddress serverAddress = InetAddress.getByName("localhost");
        Scanner userInput = new Scanner(System.in);
        byte[] sendBuffer;
        byte[] receiveBuffer = new byte[1024];
        System.out.print("Enter number of frames to send: ");
        int totalFrames = userInput.nextInt();
        userInput.nextLine();
        for (int frameIndex = 0; frameIndex < totalFrames; ) {
            System.out.print("Enter frame number: ");
            String frameNumber = userInput.nextLine();
            System.out.print("Enter message for frame " + frameNumber + ": ");
            String message = userInput.nextLine();
            String fullFrame = "Frame " + frameNumber + ": " + message;
            sendBuffer = fullFrame.getBytes();
            DatagramPacket sendPacket = new DatagramPacket(sendBuffer,
sendBuffer.length, serverAddress, 9876);
            DatagramPacket ackPacket = new DatagramPacket(receiveBuffer,
receiveBuffer.length);
            boolean isAckReceived = false;
            while (!isAckReceived) {
                clientSocket.send(sendPacket);
                System.out.println("Sent --> " + fullFrame);
```

```
            try {
                clientSocket.receive(ackPacket);
                String ackMessage = new String(ackPacket.getData(), 0,
ackPacket.getLength());
                System.out.println("ACK received --> " + ackMessage);
                isAckReceived = true;
                frameIndex++; // Move to next frame
            } catch (SocketTimeoutException e) {
                System.out.println("Timeout! No ACK for frame " + frameNumber +
". Resending...");
            }
        }
    }
    clientSocket.close();
  }
}
```

Server:

```
import java.net.*;
import java.util.Random;
public class StopAndWaitServer {
    public static void main(String[] args) throws Exception {
        DatagramSocket serverSocket = new DatagramSocket(9876);
        byte[] buffer = new byte[1024];
        Random random = new Random();
        System.out.println("Server is running and waiting for frames...");
        while (true) {
            DatagramPacket incomingPacket = new DatagramPacket(buffer,
buffer.length);
            serverSocket.receive(incomingPacket);
            String receivedFrame = new String(incomingPacket.getData(), 0,
incomingPacket.getLength());
            System.out.println("Received: " + receivedFrame);
            boolean shouldSendAck = random.nextBoolean();
            if (shouldSendAck) {
                String ackMessage = "ACK";
                byte[] ackBuffer = ackMessage.getBytes();
                DatagramPacket ackPacket = new DatagramPacket(
                    ackBuffer,
                    ackBuffer.length,
                    incomingPacket.getAddress(),
                    incomingPacket.getPort()
                );
                serverSocket.send(ackPacket);
                System.out.println("ACK sent for: " + receivedFrame);
            } else {
                System.out.println("Simulated ACK loss for: " + receivedFrame);
            }
        }
    }
}
```

Output:


C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.26100.3624]
(c) Microsoft Corporation. All rights reserved.
C:\Users\HEMANTH\Downloads\new hemanth file>javac StopAndWaitServer.java
C:\Users\HEMANTH\Downloads\new hemanth file>java StopAndWaitServer
Server is running and waiting for frames...
Received: Frame 1: cn
ACK sent for: Frame 1: cn
Received: Frame 2: lab
Simulated ACK loss for: Frame 2: lab
Received: Frame 2: lab
Simulated ACK loss for: Frame 2: lab
Received: Frame 2: lab
ACK sent for: Frame 2: lab
Received: Frame 3: tcp
Simulated ACK loss for: Frame 3: tcp
Received: Frame 3: tcp
ACK sent for: Frame 3: tcp
Received: Frame 4: udp
ACK sent for: Frame 4: udp
C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.26100.3624]
(c) Microsoft Corporation. All rights reserved.
C:\Users\HEMANTH\Downloads\new hemanth file>javac StopAndWaitClient.java
C:\Users\HEMANTH\Downloads\new hemanth file>java StopAndWaitClient
Enter number of frames to send: 4
Enter frame number: 1
Enter message for frame 1: cn
Sent --> Frame 1: cn
ACK received --> ACK
Enter frame number: 2
Enter message for frame 2: lab
Sent --> Frame 2: lab
Timeout! No ACK for frame 2. Resending...
Sent --> Frame 2: lab
Timeout! No ACK for frame 2. Resending...
Sent --> Frame 2: lab
ACK received --> ACK
Enter frame number: 3
Enter message for frame 3: tcp
Sent --> Frame 3: tcp
Timeout! No ACK for frame 3. Resending...
Sent --> Frame 3: tcp
ACK received --> ACK
Enter frame number: 4
Enter message for frame 4: udp
Sent --> Frame 4: udp
ACK received --> ACK
C:\Users\HEMANTH\Downloads\new hemanth file>

Experiment-6

AIM: To implement the dijkstra's algorithm.

Description: Dijkstra's Algorithm is used to find the **shortest path** from a source node to all other nodes in a weighted graph with non-negative edges. It works by selecting the nearest unvisited node, updating the distances to its neighbors, and repeating the process until all nodes are visited.

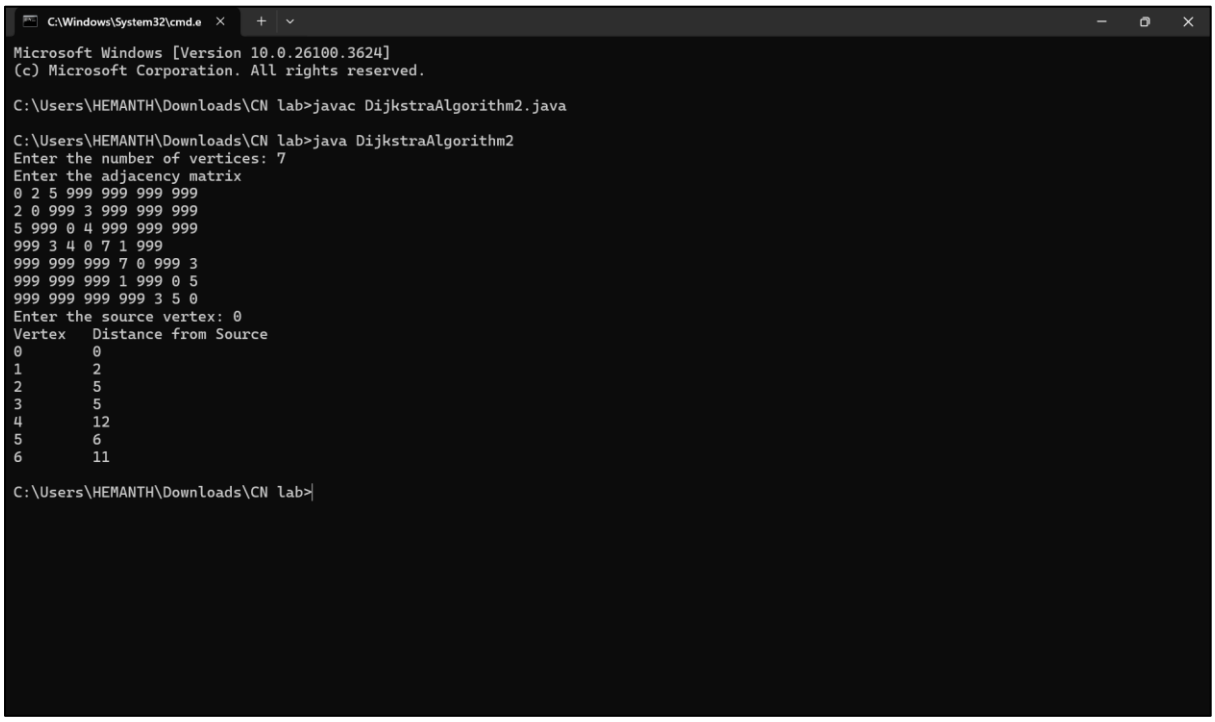
Code:

```
import java.util.*;
public class DijkstraAlgorithm2 {
   private static final int INF = 999;
   public static void dijkstra(int[][] graph, int src) {
      int vertices = graph.length;
      int[] dist = new int[vertices];
      boolean[] visited = new boolean[vertices];
      Arrays.fill(dist, INF);
      dist[src] = 0;
      for (int count = 0; count < vertices - 1; count++) {
         int u = minDistance(dist, visited, vertices);
         visited[u] = true;
         for (int v = 0; v < vertices; v++) {
            if (!visited[v] && graph[u][v] != INF && dist[u] != INF && dist[u] + graph[u][v] < dist[v])
{
               dist[v] = dist[u] + graph[u][v];
            }
         }
      }
      printSolution(dist);
   }
   private static int minDistance(int[] dist, boolean[] visited, int vertices) {
      int min = INF, minIndex = -1;
      for (int v = 0; v < vertices; v++) {
         if (!visited[v] && dist[v] < min) {
            min = dist[v];
            minIndex = v;
         }
      }
      return minIndex;
```

```
    }
    private static void printSolution(int[] dist) {
        System.out.println("Vertex              Distance from Source");
        for (int i = 0; i < dist.length; i++) {
            System.out.println(i + "    " + (dist[i] == INF ? "INF" : dist[i]));
        }
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of vertices: ");
        int vertices = scanner.nextInt();
        int[][] graph = new int[vertices][vertices];

        System.out.println("Enter the adjacency matrix");
        for (int i = 0; i < vertices; i++) {
            for (int j = 0; j < vertices; j++) {
                graph[i][j] = scanner.nextInt();
            }
        }
        System.out.print("Enter the source vertex: ");
        int src = scanner.nextInt();
        dijkstra(graph, src);
        scanner.close();
    }
}
```

Output:


C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.26100.3624]
(c) Microsoft Corporation. All rights reserved.
C:\Users\HEMANTH\Downloads\CN lab>javac DijkstraAlgorithm2.java
C:\Users\HEMANTH\Downloads\CN lab>java DijkstraAlgorithm2
Enter the number of vertices: 7
Enter the adjacency matrix
0 2 5 999 999 999 999
2 0 999 3 999 999 999
5 999 0 4 999 999 999
999 3 4 0 7 1 999
999 999 999 7 0 999 3
999 999 999 1 999 0 5
999 999 999 999 3 5 0
Enter the source vertex: 0
Vertex Distance from Source
0 0
1 2
2 5
3 5
4 12
5 6
6 11
C:\Users\HEMANTH\Downloads\CN lab>

Experiment-7

AIM: To implement a DistanceVectorRouting program.

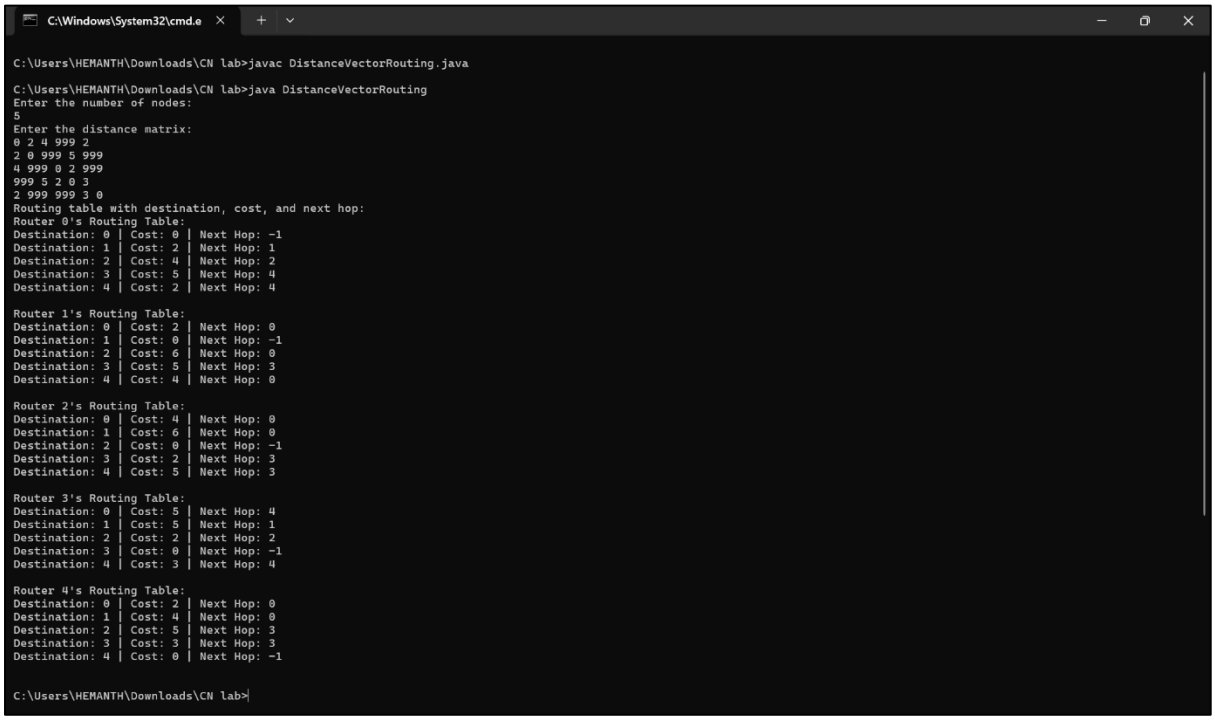
Description: Distance Vector Routing is a routing protocol where each router shares its routing table with its neighbors. Routers update their tables based on the shortest path (least cost) to each destination, using the Bellman-Ford algorithm. Updates happen periodically or when there are changes in the network.

Code:

```
import java.util.*;
class DistanceVectorRouting {
    public static void main(String[] args) {
        int totalNodes;
        Scanner inputScanner = new Scanner(System.in);
        System.out.println("Enter the number of nodes: ");
        totalNodes = inputScanner.nextInt();
        int distanceMatrix[][] = new int[totalNodes][totalNodes];
        System.out.println("Enter the distance matrix: ");
        for (int x = 0; x < totalNodes; x++) {
            for (int y = 0; y < totalNodes; y++) {
                distanceMatrix[x][y] = inputScanner.nextInt();
            }
        }
        int[][] routingData = new int[totalNodes][totalNodes];
        int[][] nextHop = new int[totalNodes][totalNodes];
        for (int x = 0; x < totalNodes; x++) {
            for (int y = 0; y < totalNodes; y++) {
                if (distanceMatrix[x][y] == 0) {
                    if (x == y) {
                        routingData[x][y] = 0;
                    } else {
                        routingData[x][y] = 999;
                    }
                    nextHop[x][y] = -1;
                } else {
                    routingData[x][y] = distanceMatrix[x][y]; // Direct link cost
                    nextHop[x][y] = y; // Direct link, next hop is destination itself
                }
```

```
            }
        }
        boolean tableUpdated;
        do {
            tableUpdated = false;
            for (int x = 0; x < totalNodes; x++) {
                for (int y = 0; y < totalNodes; y++) {
                    if (x != y) { // Ignore the case where the source and destination are the same
                        for (int z = 0; z < totalNodes; z++) {
                            if (routingData[x][z] + routingData[z][y] < routingData[x][y]) {
                                routingData[x][y] = routingData[x][z] + routingData[z][y];
                                nextHop[x][y] = nextHop[x][z];
                                tableUpdated = true;
                            }
                        }
                    }
                }
            }
        } while (tableUpdated);
        System.out.println("Routing table with destination, cost, and next hop: ");
        for (int x = 0; x < totalNodes; x++) {
            System.out.println("Router " + x + "'s Routing Table:");
            for (int y = 0; y < totalNodes; y++) {
                if (routingData[x][y] == 999) {
                    System.out.println("Destination: " + y + " | Cost: 999 | Next Hop: None");
                } else {
                    System.out.println("Destination: " + y + " | Cost: " + routingData[x][y] + " | Next Hop: " +
nextHop[x][y]);
                }
            }
            System.out.println();
        }
        inputScanner.close();
    }
}
```

Output:


C:\Windows\System32\cmd.e
C:\Users\HEMANTH\Downloads\CN lab>javac DistanceVectorRouting.java
C:\Users\HEMANTH\Downloads\CN lab>java DistanceVectorRouting
Enter the number of nodes:
5
Enter the distance matrix:
0 2 4 999 2
2 0 999 5 999
4 999 0 2 999
999 5 2 0 3
2 999 999 3 0
Routing table with destination, cost, and next hop:
Router 0's Routing Table:
Destination: 0 | Cost: 0 | Next Hop: -1
Destination: 1 | Cost: 2 | Next Hop: 1
Destination: 2 | Cost: 4 | Next Hop: 2
Destination: 3 | Cost: 5 | Next Hop: 4
Destination: 4 | Cost: 2 | Next Hop: 4
Router 1's Routing Table:
Destination: 0 | Cost: 2 | Next Hop: 0
Destination: 1 | Cost: 0 | Next Hop: -1
Destination: 2 | Cost: 6 | Next Hop: 0
Destination: 3 | Cost: 5 | Next Hop: 3
Destination: 4 | Cost: 4 | Next Hop: 0
Router 2's Routing Table:
Destination: 0 | Cost: 4 | Next Hop: 0
Destination: 1 | Cost: 6 | Next Hop: 0
Destination: 2 | Cost: 0 | Next Hop: -1
Destination: 3 | Cost: 2 | Next Hop: 3
Destination: 4 | Cost: 5 | Next Hop: 3
Router 3's Routing Table:
Destination: 0 | Cost: 5 | Next Hop: 4
Destination: 1 | Cost: 5 | Next Hop: 1
Destination: 2 | Cost: 2 | Next Hop: 2
Destination: 3 | Cost: 0 | Next Hop: -1
Destination: 4 | Cost: 3 | Next Hop: 4
Router 4's Routing Table:
Destination: 0 | Cost: 2 | Next Hop: 0
Destination: 1 | Cost: 4 | Next Hop: 0
Destination: 2 | Cost: 5 | Next Hop: 3
Destination: 3 | Cost: 3 | Next Hop: 3
Destination: 4 | Cost: 0 | Next Hop: -1
C:\Users\HEMANTH\Downloads\CN lab>

Experiment-8

AIM: To implement a program on leakybucket algorithm.

Description: The Leaky Bucket Algorithm is used for **traffic shaping** and **rate control** in networks. Data packets are added to a bucket (buffer) and sent out at a **fixed rate**, like water leaking from a bucket. If the bucket overflows (too much data too fast), excess packets are discarded, helping prevent congestion.

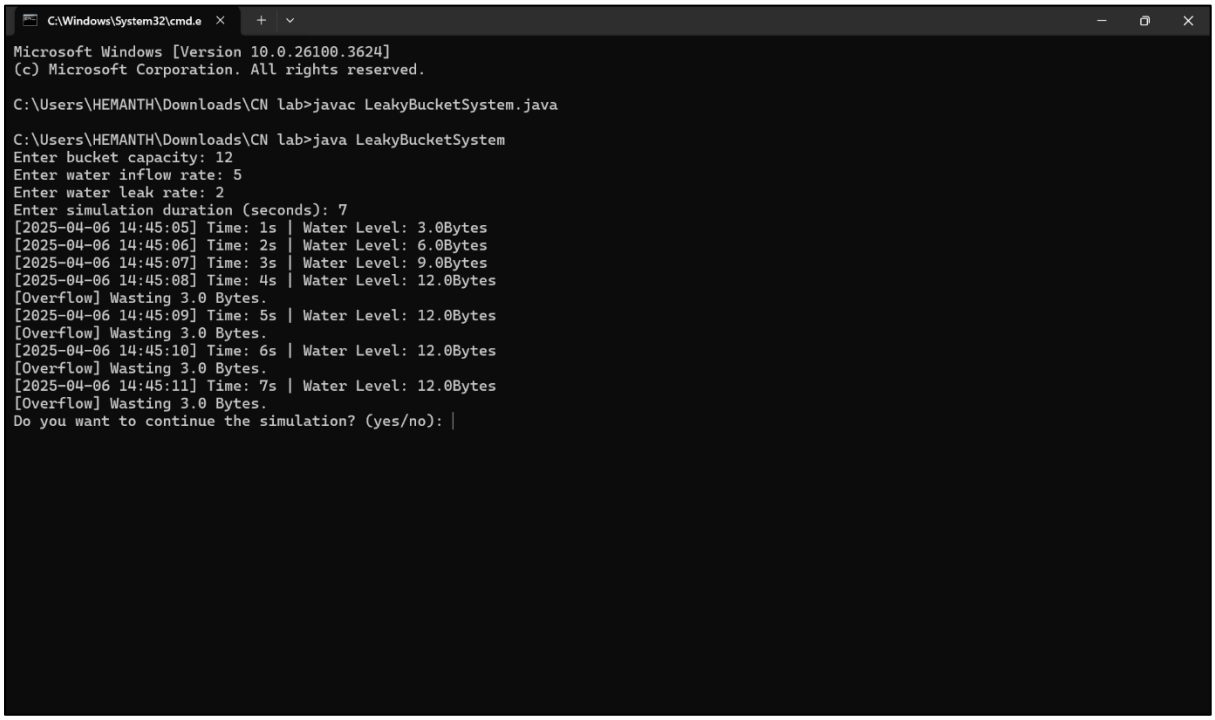
Code:

```
import java.util.Scanner;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
class LeakyBucketSystem {
    private final double capacity;
    private double currentWater;
    private final double fillRate;
    private final double leakRate;
    private boolean running = true;
    public LeakyBucketSystem(double capacity, double fillRate, double leakRate) {
        this.capacity = capacity;
        this.fillRate = fillRate;
        this.leakRate = leakRate;
        this.currentWater = 0;
    }
    public void startSimulation(int duration) {
        Thread fillThread = new Thread(this::fill);
        Thread leakThread = new Thread(this::leak);
        Thread monitorThread = new Thread(() -> monitor(duration));
        fillThread.start();
        leakThread.start();
        monitorThread.start();
        try {
            monitorThread.join();
            running = false;
            fillThread.join();
            leakThread.join();
        } catch (InterruptedException e) {
            System.out.println("Simulation interrupted.");
```

```
        }
    }
    private void fill() {
        while (running) {
            synchronized (this) {
                currentWater += fillRate;
                if (currentWater > capacity) {
                    System.out.println("[Overflow] Wasting " + (currentWater - capacity) + " Bytes.");
                    currentWater = capacity;
                }
            }
            sleep(1000);
        }
    }
    private void leak() {
        while (running) {
            synchronized (this) {
                currentWater -= leakRate;
                if (currentWater < 0) {
                    currentWater = 0;
                }
            }
            sleep(1000);
        }
    }
    private void monitor(int duration) {
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
        for (int i = 0; i < duration; i++) {
            synchronized (this) {
                String formattedTime = LocalDateTime.now().format(formatter);
                System.out.println("[" + formattedTime + "] Time: " + (i + 1) + "s | Water Level: " +
currentWater + "Bytes");
            }
            sleep(1000);
        }
    }
    private void sleep(int millis) {
```

```
        try {
            Thread.sleep(millis);
        } catch (InterruptedException e) {
            System.out.println("Thread interrupted.");
        }
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter bucket capacity: ");
        double capacity = scanner.nextDouble();
        System.out.print("Enter water inflow rate: ");
        double fillRate = scanner.nextDouble();
        System.out.print("Enter water leak rate: ");
        double leakRate = scanner.nextDouble();
        LeakyBucketSystem bucket = new LeakyBucketSystem(capacity, fillRate, leakRate);
        while (true) {
            System.out.print("Enter simulation duration (seconds): ");
            int duration = scanner.nextInt();
            bucket.startSimulation(duration);
            System.out.print("Do you want to continue the simulation? (yes/no): ");
            String choice = scanner.next();
            if (!choice.equalsIgnoreCase("yes")) {
                break;
            }
        }
        scanner.close();
    }
}
```

Output:


C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.26100.3624]
(c) Microsoft Corporation. All rights reserved.
C:\Users\HEMANTH\Downloads\CN lab>javac LeakyBucketSystem.java
C:\Users\HEMANTH\Downloads\CN lab>java LeakyBucketSystem
Enter bucket capacity: 12
Enter water inflow rate: 5
Enter water leak rate: 2
Enter simulation duration (seconds): 7
[2025-04-06 14:45:05] Time: 1s | Water Level: 3.0Bytes
[2025-04-06 14:45:06] Time: 2s | Water Level: 6.0Bytes
[2025-04-06 14:45:07] Time: 3s | Water Level: 9.0Bytes
[2025-04-06 14:45:08] Time: 4s | Water Level: 12.0Bytes
[Overflow] Wasting 3.0 Bytes.
[2025-04-06 14:45:09] Time: 5s | Water Level: 12.0Bytes
[Overflow] Wasting 3.0 Bytes.
[2025-04-06 14:45:10] Time: 6s | Water Level: 12.0Bytes
[Overflow] Wasting 3.0 Bytes.
[2025-04-06 14:45:11] Time: 7s | Water Level: 12.0Bytes
[Overflow] Wasting 3.0 Bytes.
Do you want to continue the simulation? (yes/no):

Experiment-9

AIM: To implement Tcp client and server program.

Description: In a TCP connection, the **server** listens for incoming requests on a specific port, while the **client** initiates the connection. TCP (Transmission Control Protocol) ensures reliable, ordered, and error-free communication between the two. Once connected, they can exchange data until the connection is closed.

Client:

```
import java.io.*;
import java.net.*;
class TCPClient {
    public static void main(String[] args) {
        String serverHost = "localhost";
        int serverPort = 7000;
        try (Socket clientSocket = new Socket(serverHost, serverPort);
            BufferedReader serverInput =new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            PrintWriter serverOutput = new PrintWriter(clientSocket.getOutputStream(), true);
            BufferedReader userConsoleInput =new BufferedReader(new
InputStreamReader(System.in))) {
            System.out.println(serverInput.readLine());
            System.out.println("Connected to server. Type messages (type 'exit' to quit):");
            String userMessage;
            while (true) {
                System.out.print("You: ");
                userMessage = userConsoleInput.readLine();
                serverOutput.println(userMessage);

                if ("exit".equalsIgnoreCase(userMessage)) {
                    break;
                }
                String serverResponse = serverInput.readLine();
                System.out.println(serverResponse);
            }
            System.out.println("Connection closed.");
        } catch (IOException ex) {
            ex.printStackTrace();
```

```
        }
    }
}
```

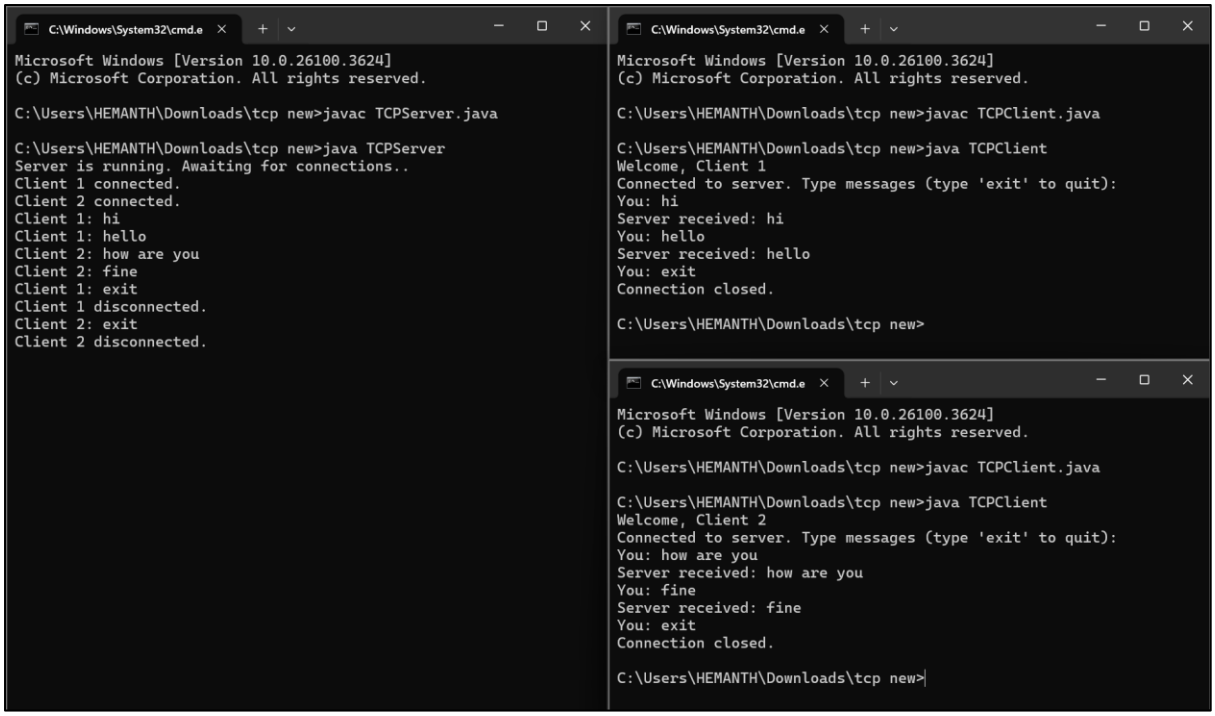
Server:

```
import java.io.*;
import java.net.*;
class TCPServer {
    private static final int SERVER_PORT = 7000;
    private static int clientCount = 0;
    public static void main(String[] args) {
        try (ServerSocket serverListener = new ServerSocket(SERVER_PORT)) {
            System.out.println("Server is running. Awaiting for connections..");
            while (true) {
                Socket clientSocket = serverListener.accept();
                clientCount++;
                System.out.println("Client " + clientCount + " connected.");
                new ClientHandlerThread(clientSocket, clientCount).start();
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
class ClientHandlerThread extends Thread {
    private Socket clientSocket;
    private int clientID;

    public ClientHandlerThread(Socket clientSocket, int clientID) {
        this.clientSocket = clientSocket;
        this.clientID = clientID;
    }
    public void run() {
        try (
            BufferedReader reader = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            PrintWriter writer = new PrintWriter(clientSocket.getOutputStream(), true)
        ) {
            writer.println("Welcome, Client " + clientID + " ");
            String clientInput;
            while ((clientInput = reader.readLine()) != null) {
```

```
            System.out.println("Client " + clientID + ": " + clientInput);
            writer.println("Server received: " + clientInput);
            if (clientInput.equalsIgnoreCase("exit")) {
                break;
            }
        }
        clientSocket.close();
        System.out.println("Client " + clientID + " disconnected.");
    } catch (IOException ex) {
        ex.printStackTrace();
    }
  }
}
```

Output:



Experiment-10

AIM: To implement udp client and server program.

Description: In a UDP (User Datagram Protocol) connection, the server listens on a port and the client sends messages (datagrams) to it. Unlike TCP, UDP is connectionless, faster, and does not guarantee delivery, order, or error checking—making it suitable for real-time applications like video streaming or gaming.

Client:

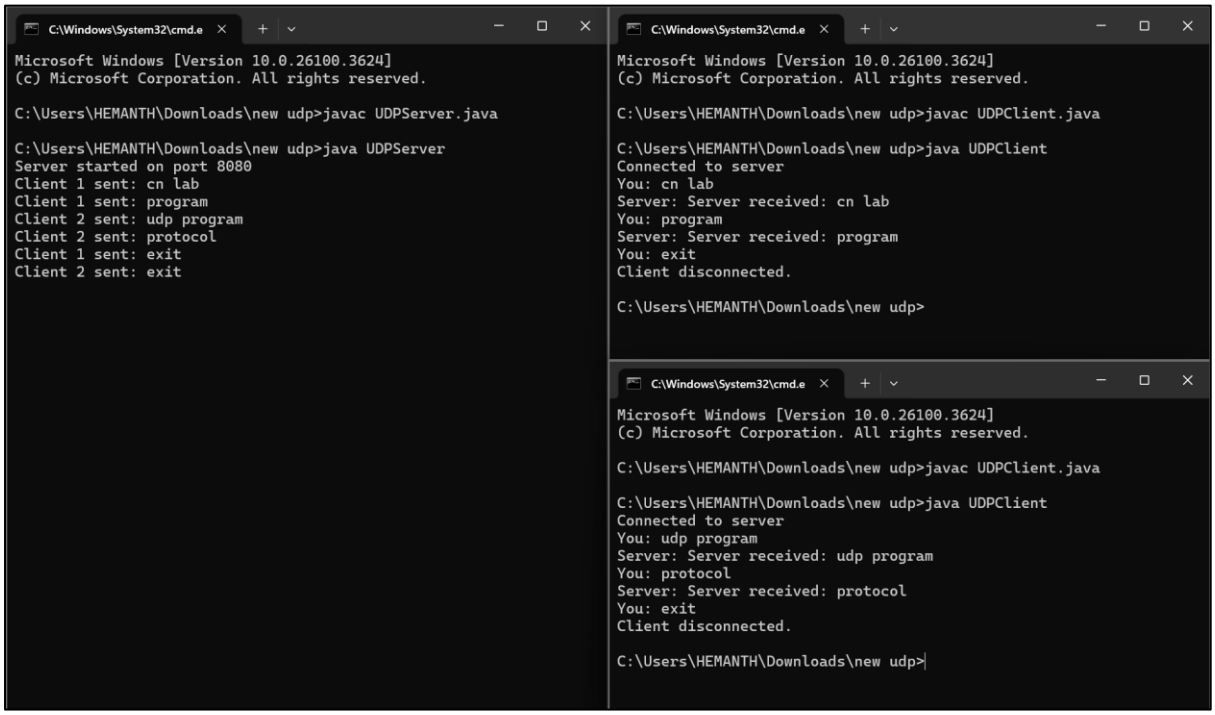
```
import java.io.*;
import java.net.*;
public class UDPClient {
    private static final String HOST = "localhost";
    private static final int PORT = 8080;
    public static void main(String[] args) {
        try (DatagramSocket udpSocket = new DatagramSocket();
             BufferedReader consoleInput = new BufferedReader(new InputStreamReader(System.in))) {
            InetAddress serverHost = InetAddress.getByName(HOST);
            byte[] receiveBuffer = new byte[1024];
            System.out.println("Connected to server");
            while (true) {
                System.out.print("You: ");
                String userMessage = consoleInput.readLine();
                DatagramPacket sendPacket = new DatagramPacket(userMessage.getBytes(),
userMessage.length(), serverHost, PORT);
                udpSocket.send(sendPacket);
                if ("exit".equalsIgnoreCase(userMessage)) {
                    System.out.println("Client disconnected.");
                    break;
                }
                DatagramPacket responsePacket = new DatagramPacket(receiveBuffer,
receiveBuffer.length);
                udpSocket.receive(responsePacket);
                System.out.println("Server:"+new String(responsePacket.getData(), 0,
responsePacket.getLength()));
            }
        } catch (IOException ex) {
            System.out.println("Connection error: " + ex.getMessage());
```

```
        }
    }
}
```

Server:

```
import java.io.*;
import java.net.*;
import java.util.concurrent.ConcurrentHashMap;
public class UDPServer {
   private static final int SERVER_PORT = 8080;
   private static boolean isRunning = true;
   private static final ConcurrentHashMap<String, Integer> connectedClients = new ConcurrentHashMap<>();
   public static void main(String[] args) {
      try (DatagramSocket udpSocket = new DatagramSocket(SERVER_PORT)) {
         System.out.println("Server started on port " + SERVER_PORT);
         byte[] receiveBuffer = new byte[1024];
         while (isRunning) {
            DatagramPacket receivedPacket = new DatagramPacket(receiveBuffer, receiveBuffer.length);
            udpSocket.receive(receivedPacket);
            String clientKey = receivedPacket.getAddress() + ":" + receivedPacket.getPort();
            connectedClients.putIfAbsent(clientKey, connectedClients.size() + 1);
            int clientId = connectedClients.get(clientKey);
            String clientMessage = new String(receivedPacket.getData(), 0, receivedPacket.getLength());
            System.out.println("Client " + clientId + " sent: " + clientMessage);
            if ("exit".equalsIgnoreCase(clientMessage)) {
               connectedClients.remove(clientKey);
               continue;
            }
            byte[] responseBuffer = ("Server received: " + clientMessage).getBytes();
            DatagramPacket responsePacket = new DatagramPacket(responseBuffer, responseBuffer.length,
                  receivedPacket.getAddress(), receivedPacket.getPort());
            udpSocket.send(responsePacket);
         }
      } catch (IOException ex) {
         System.out.println("Server error: " + ex.getMessage());
      }
   }
}
```

Output:


C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.26100.3624]
(c) Microsoft Corporation. All rights reserved.
C:\Users\HEMANTH\Downloads\new udp>javac UDPServer.java
C:\Users\HEMANTH\Downloads\new udp>java UDPServer
Server started on port 8080
Client 1 sent: cn lab
Client 1 sent: program
Client 2 sent: udp program
Client 2 sent: protocol
Client 1 sent: exit
Client 2 sent: exit
C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.26100.3624]
(c) Microsoft Corporation. All rights reserved.
C:\Users\HEMANTH\Downloads\new udp>javac UDPClient.java
C:\Users\HEMANTH\Downloads\new udp>java UDPClient
Connected to server
You: cn lab
Server: Server received: cn lab
You: program
Server: Server received: program
You: exit
Client disconnected.
C:\Users\HEMANTH\Downloads\new udp>
C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.26100.3624]
(c) Microsoft Corporation. All rights reserved.
C:\Users\HEMANTH\Downloads\new udp>javac UDPClient.java
C:\Users\HEMANTH\Downloads\new udp>java UDPClient
Connected to server
You: udp program
Server: Server received: udp program
You: protocol
Server: Server received: protocol
You: exit
Client disconnected.
C:\Users\HEMANTH\Downloads\new udp>