EX.NO. 1 DATE: 21/08/2021	IMPLEMENTION OF THE BASIC COMMANDS

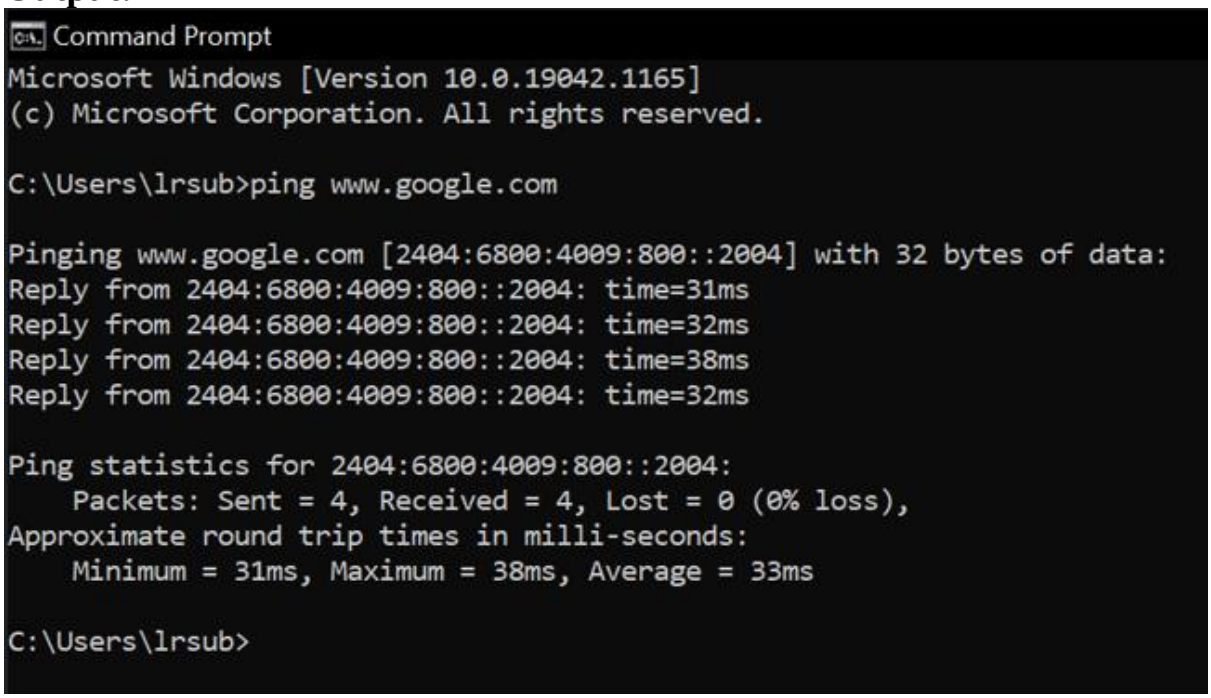
Aim:

To implement the basic commands

1) Ping command:

- The ping command is a Command Prompt command used to test the ability of the source computer to reach a specified destination computer.
- The ping command sends one datagram per second and prints one line of output for every response received.
- The ping command calculates round-trip times and packet loss statistics, and displays a brief summary on completion.
- The general format is ping hostname or ping IP address.
- Example **ping www.google.com** or **ping 216.58.208.68**.

Output:

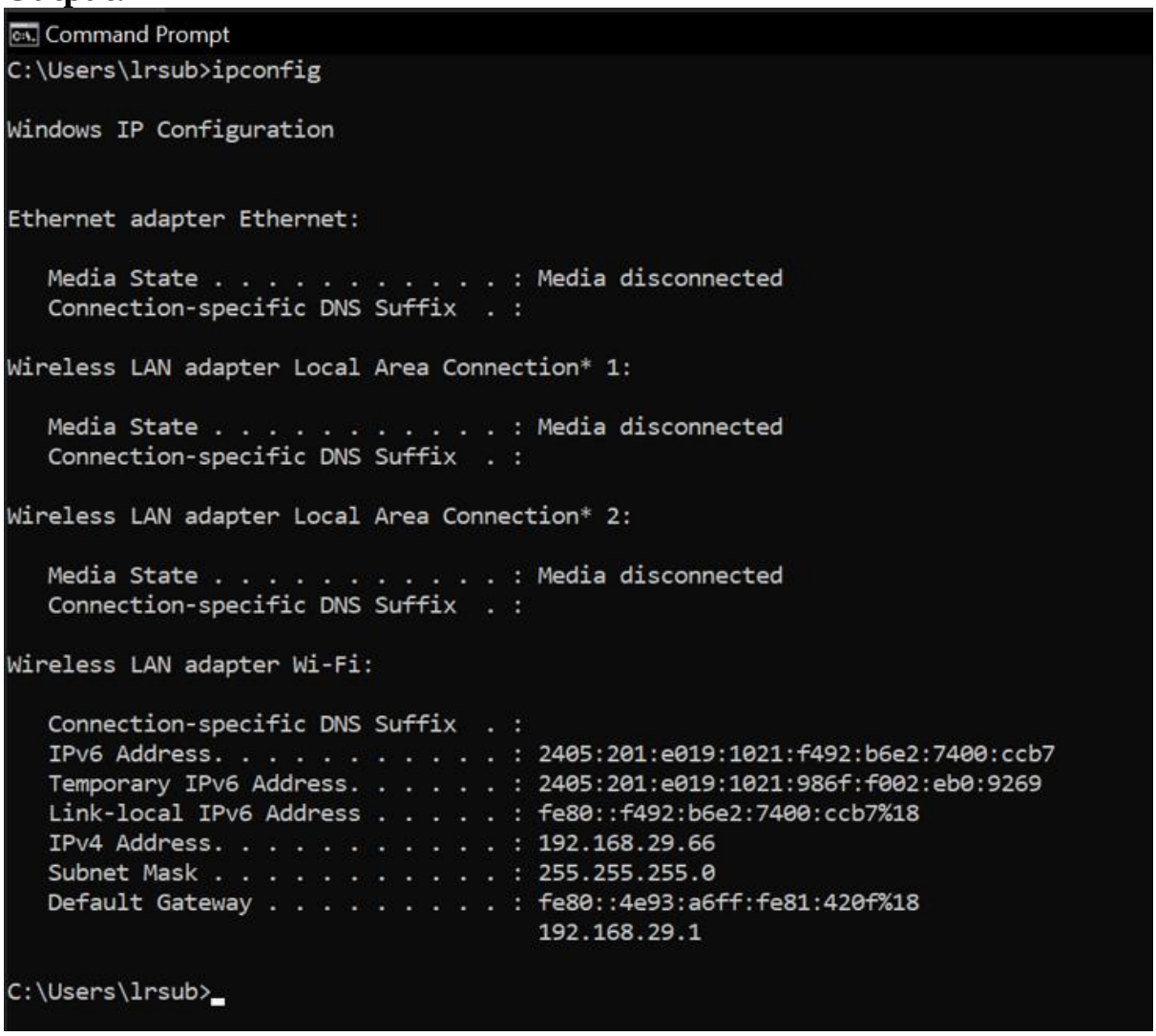


2) ipconfig command:

- ipconfig (standing for "Internet Protocol configuration") is a console application program of some computer operating systems that displays all current TCP/IP network configuration values and refreshes Dynamic Host Configuration Protocol (DHCP) and Domain Name System (DNS) settings.

- Another indispensable and frequently used utility that is used for finding network information about your local machine like IP addresses, DNS addresses etc
- Basic Use: Finding Your IP Address and Default Gateway

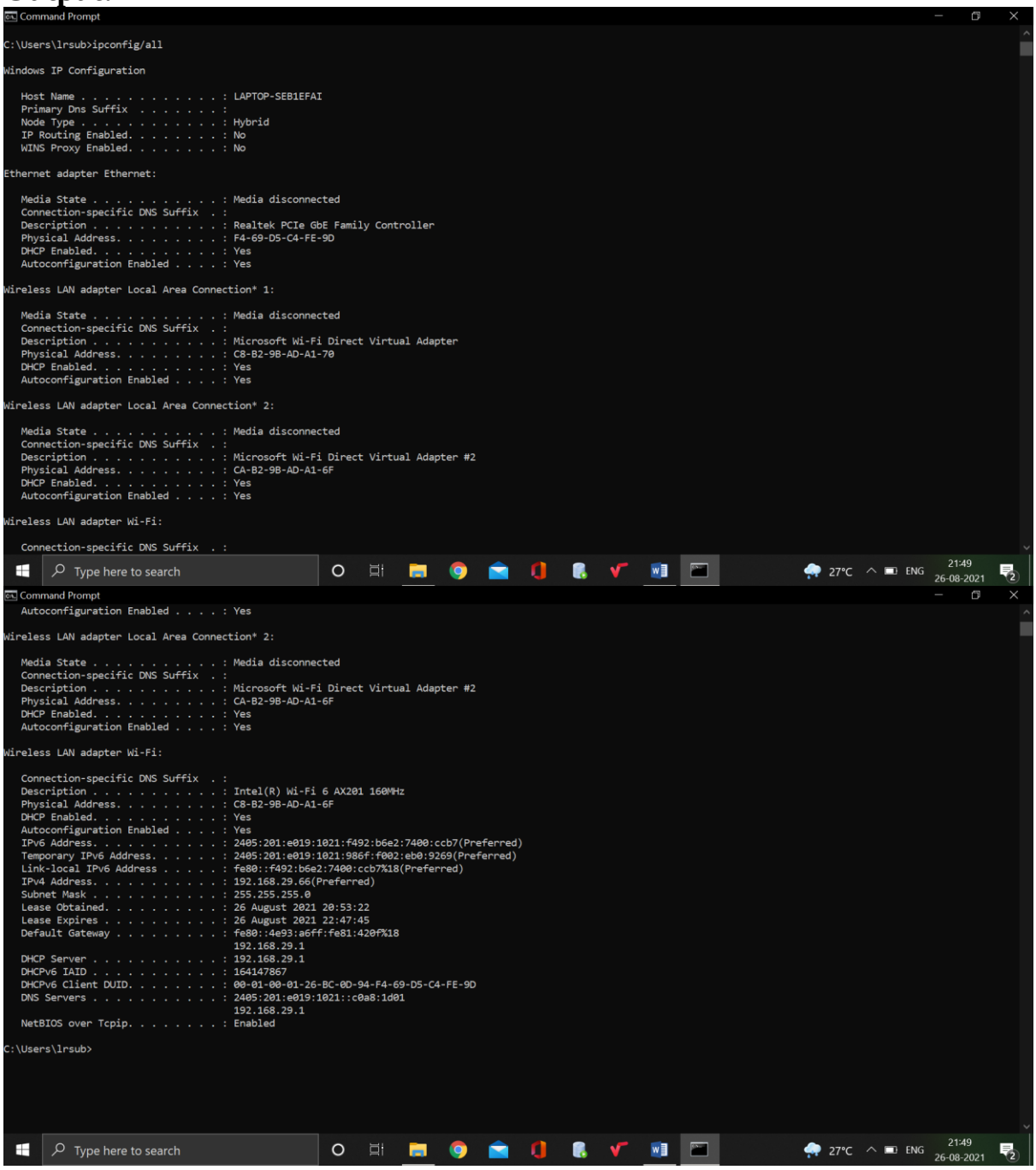
Output:



(a) ipconfig/all command:

- ipconfig /all displays all configuration information for each adapter bound to TCP/IP.
- It displays more information about the network setup on your systems including the MAC address.

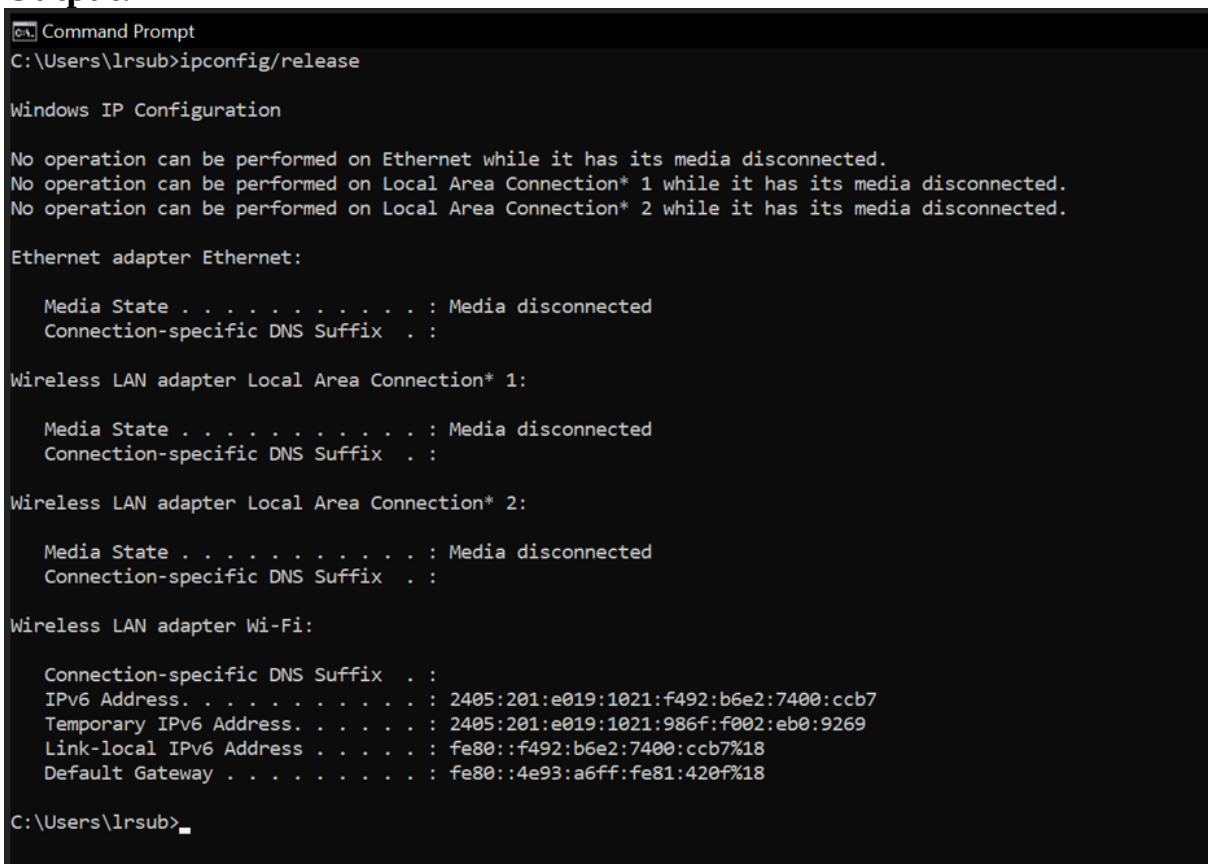
Output:



(b) ipconfig/release command:

- It releases the current IP address.
- First, ipconfig /release is executed to force the client to immediately give up its lease by sending the server a DHCP release notification which updates the server's status information and marks the old client's IP address as "available".
- Then, the command ipconfig /renew is executed to request a new IP address.

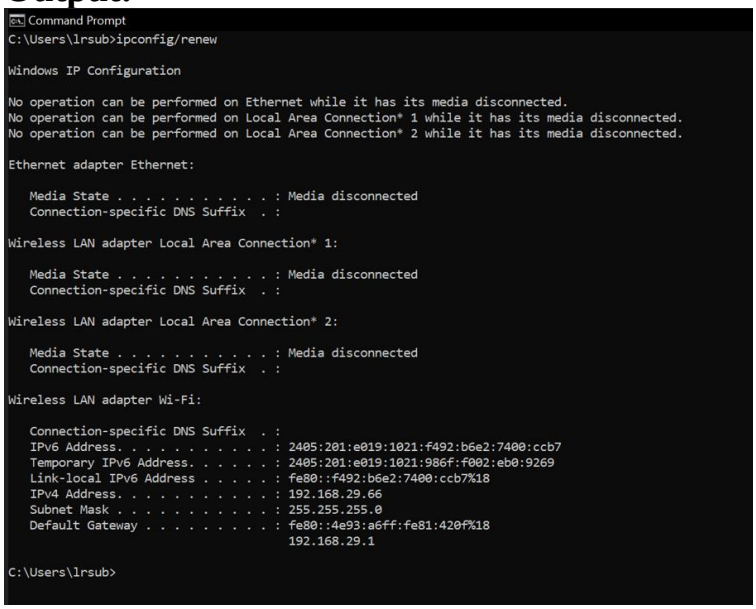
Output:



(c) ipconfig/renew command:

- It renews IP address.
- Ipconfig /renew is the command used to tell the DHCP server that your computer wishes to join the network and needs to be configured with an IP address to communicate with the other devices on the network.

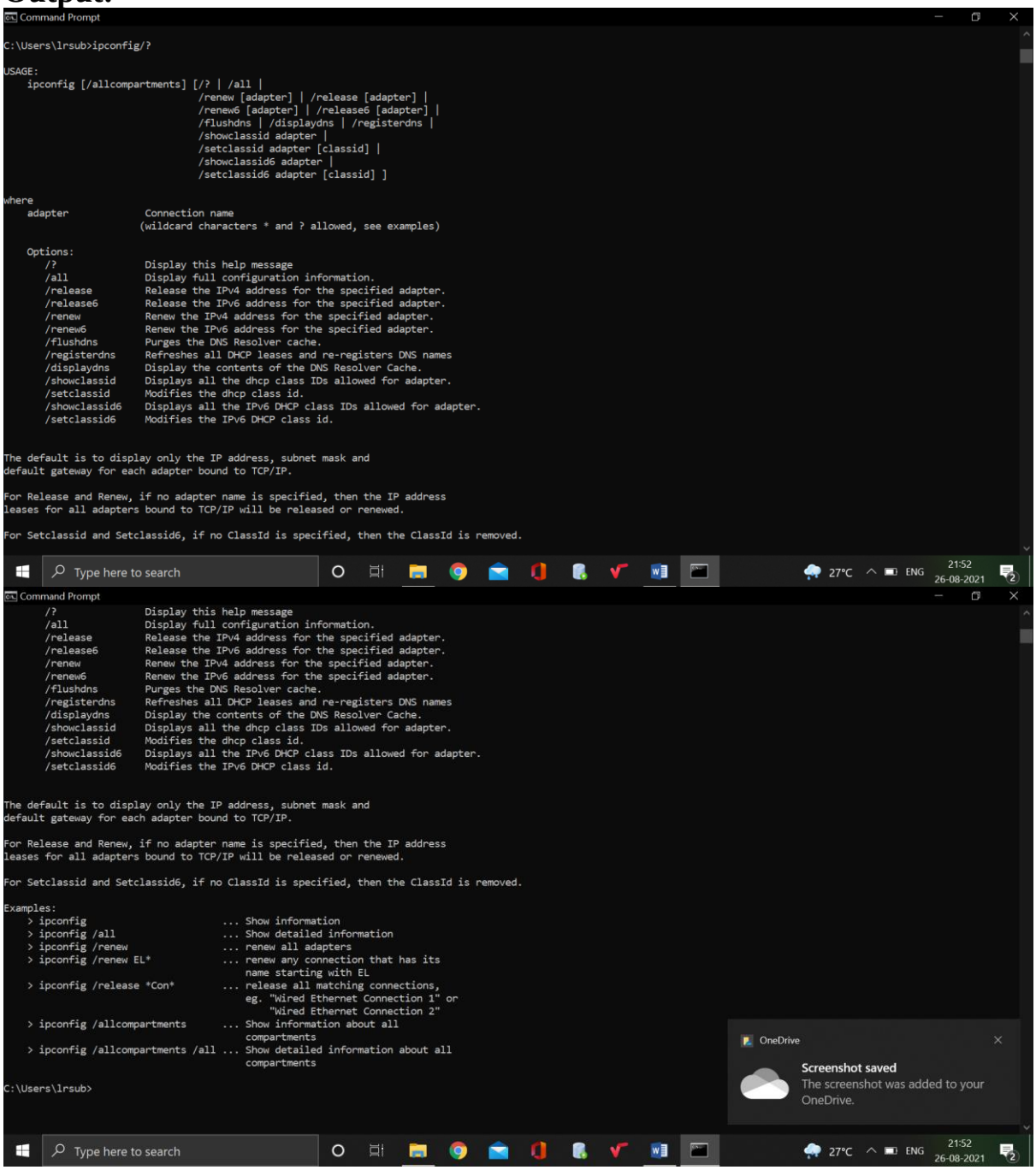
Output:



(d) ipconfig/? command:

- It shows help.

Output:

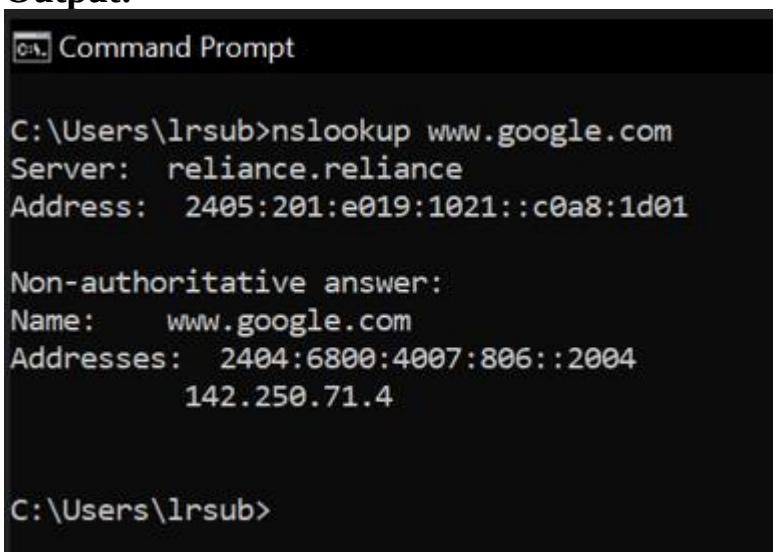


3. NSlookup command:

- Used for checking DNS record entries.
- nslookup is a network administration command-line tool available for many computer operating systems.
- It is used for querying the Domain Name System (DNS) to obtain domain name or IP address mapping information.

- The main use of nslookup is for troubleshooting DNS related problems.

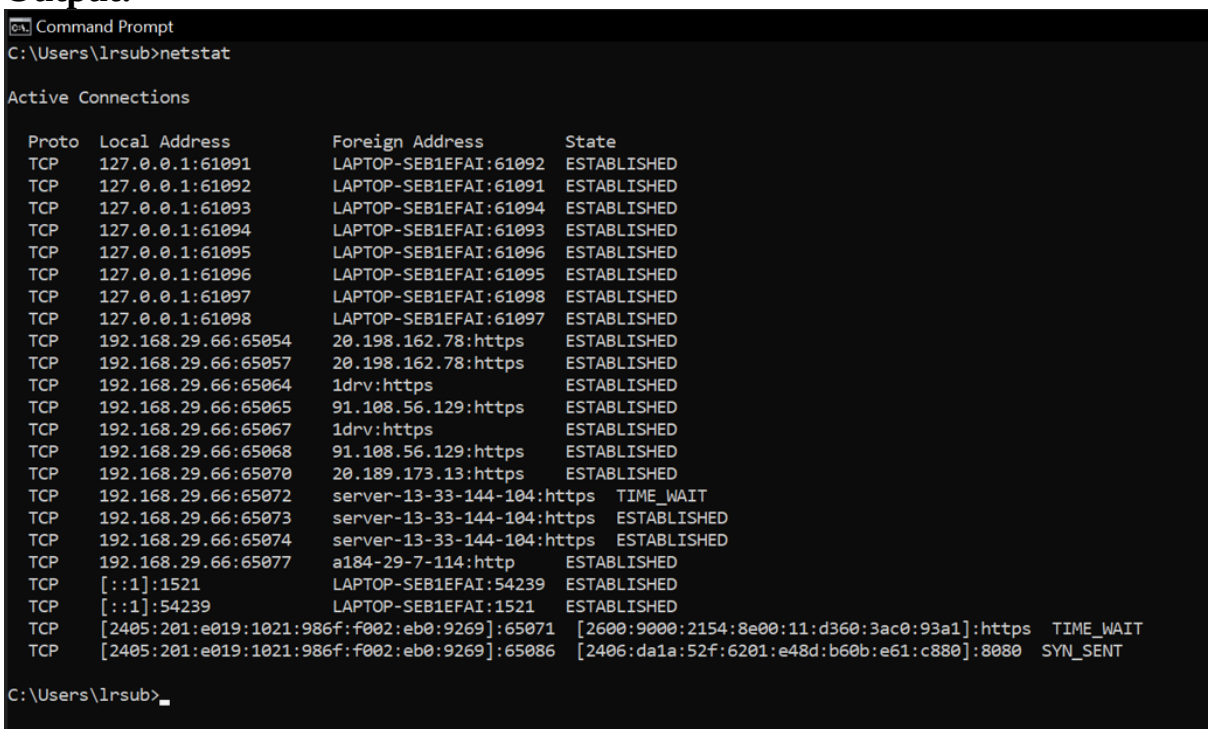
Output:



4. Netstat command:

- The netstat command generates displays that show network status and protocol statistics.
- You can display the status of TCP and UDP endpoints in table format, routing table information, and interface information.
- The most frequently used options for determining network status are: s, r, and i.

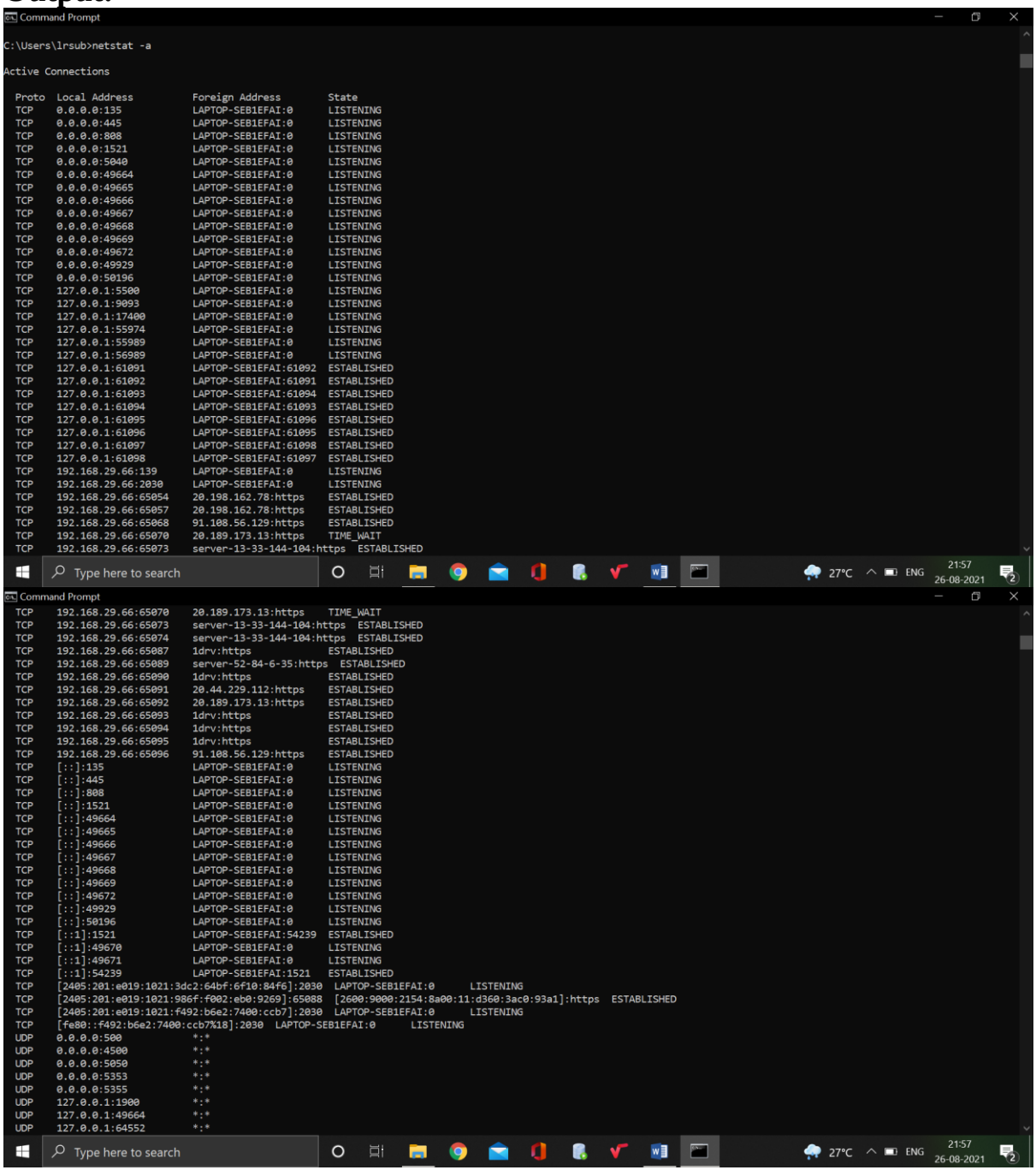
Output:

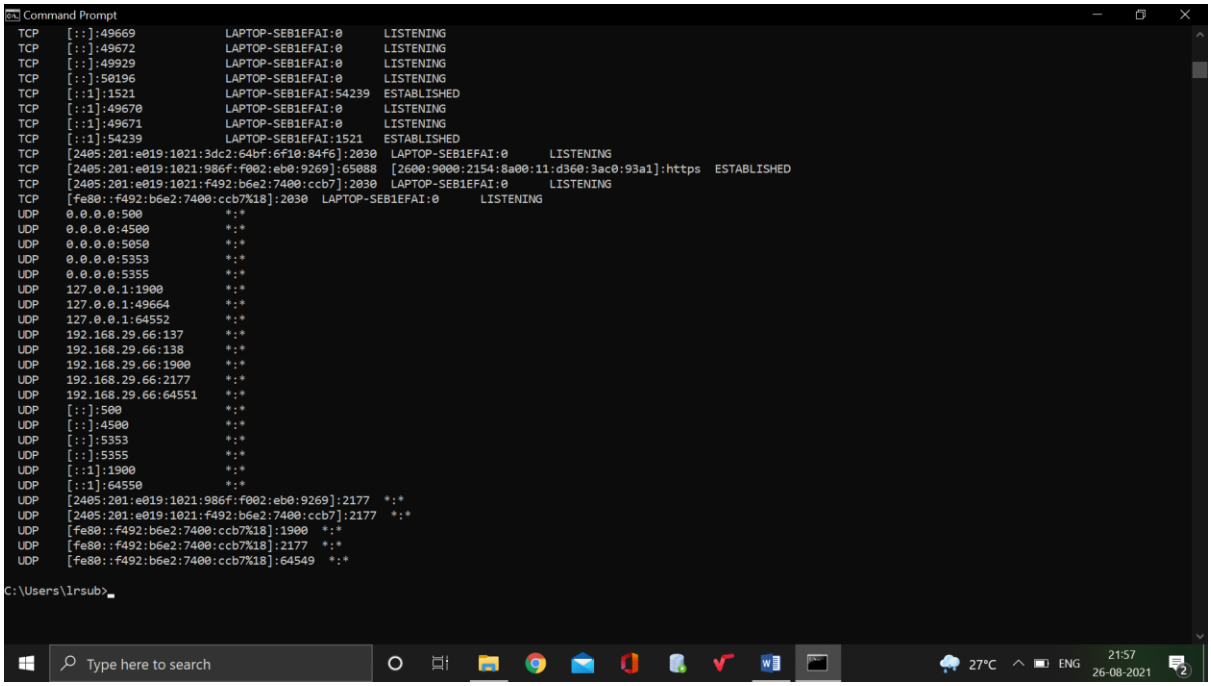


(a) netstat -a command:

- This switch displays active TCP connections, TCP connections with the listening state, as well as UDP ports that are being listened to.

Output:

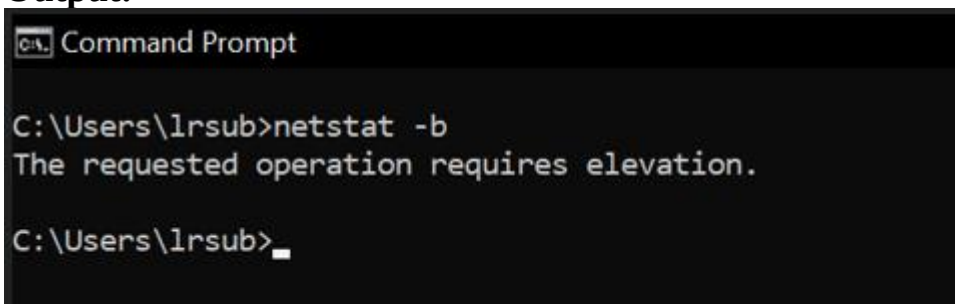




(b) netstat -b command:

- This netstat switch is very similar to the **-o** switch listed below, but instead of displaying the PID, will display the process's actual file name.
- Using **-b** over **-o** might seem like it's saving you a step or two but using it can sometimes greatly extend the time it takes netstat to fully execute.

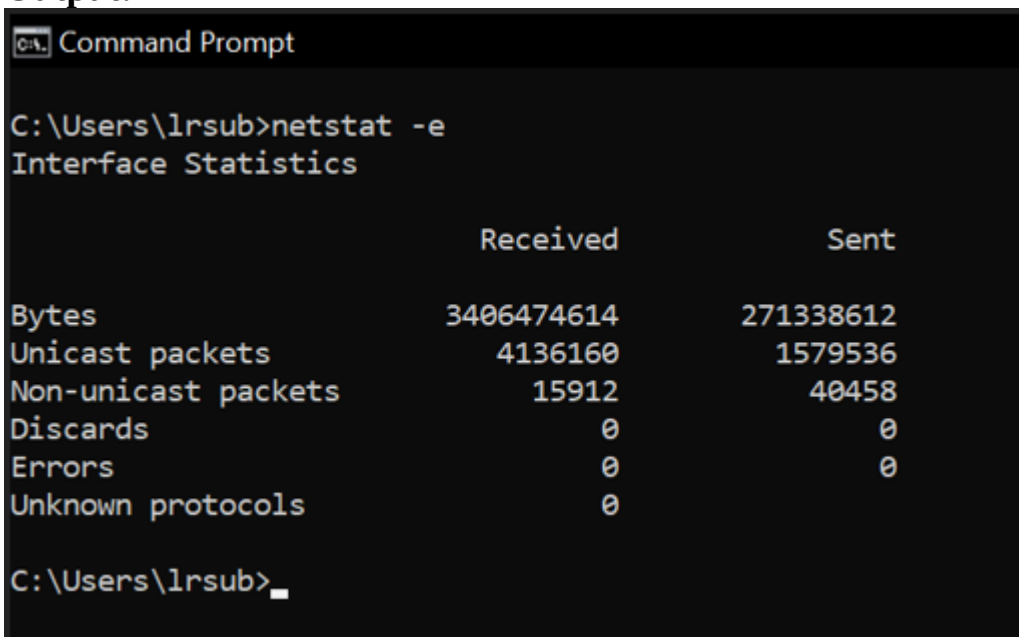
Output:



(c) netstat -e command:

- Use this switch with the netstat command to show statistics about your network connection.
- This data includes bytes, unicast packets, non-unicast packets, discards, errors, and unknown protocols received and sent since the connection was established.

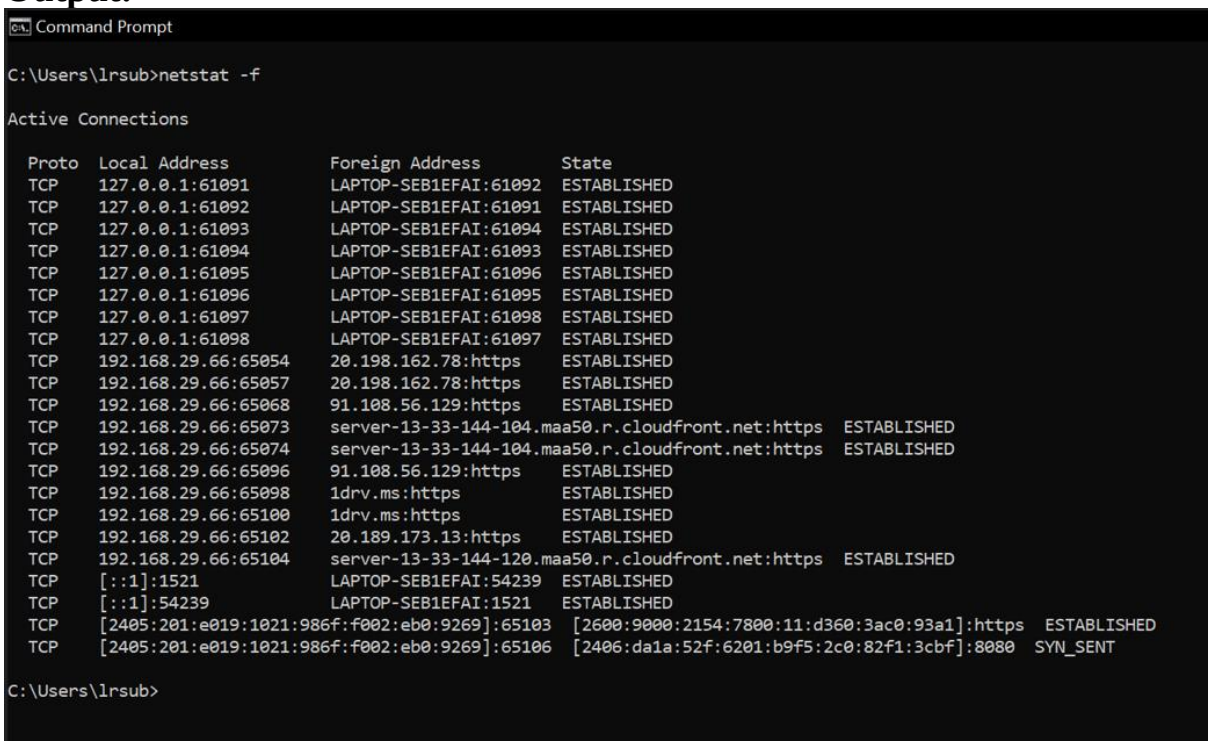
Output:



(d) netstat -f command:

- The **-f** switch will force the netstat command to display the Fully Qualified Domain Name (FQDN) for each foreign IP addresses when possible.

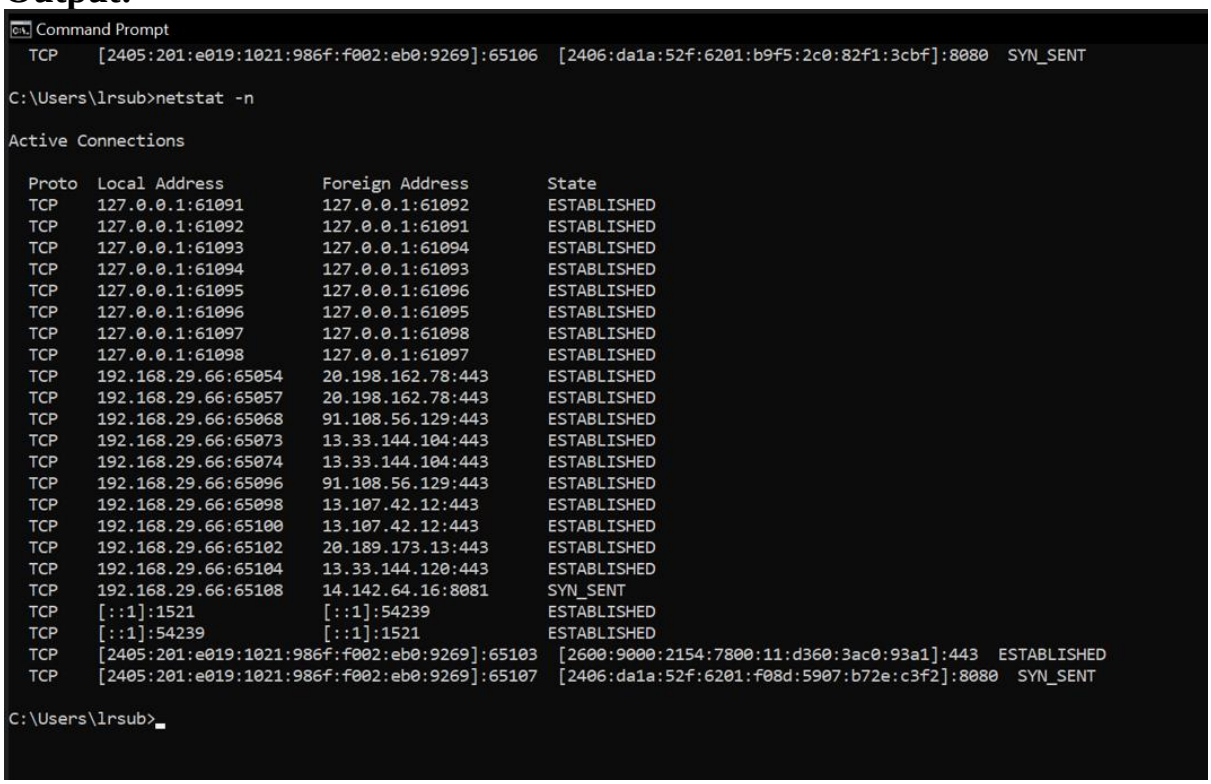
Output:



(e) netstat -n command:

- Use the **-n** switch to prevent netstat from attempting to determine host names for foreign IP addresses.
- Depending on your current network connections, using this switch could considerably reduce the time it takes for netstat to fully execute.

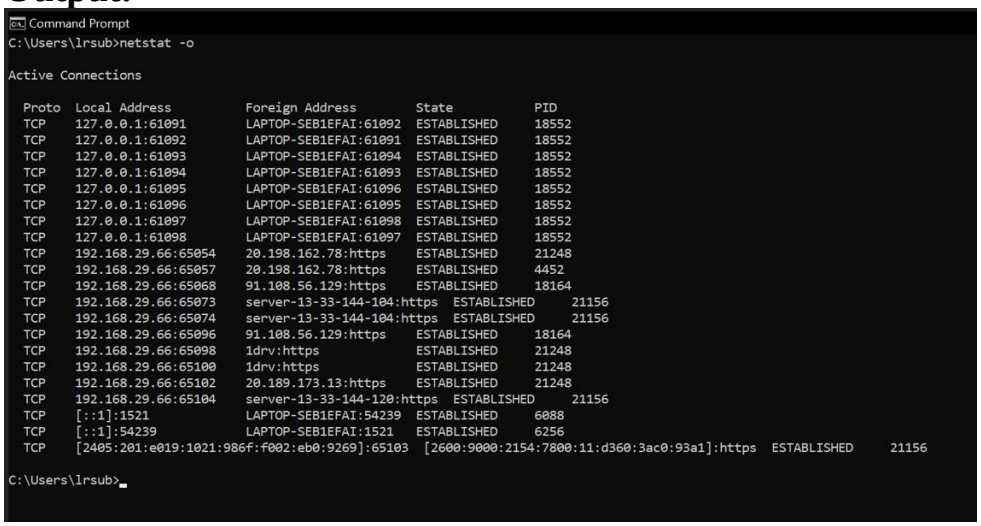
Output:



(f) netstat -o command:

- A handy option for many troubleshooting tasks, the **-o** switch displays the process identifier (PID) associated with each displayed connection.

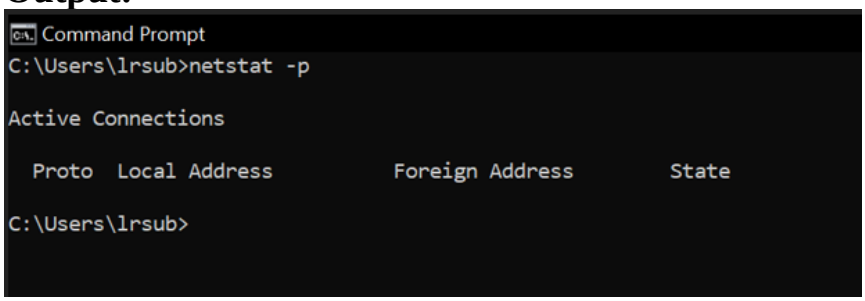
Output:



(g) netstat -p command:

- Use the **-p** switch to show connections or statistics only for a particular protocol.
- You cannot define more than one protocol at once, nor can you execute netstat with **-p** without defining a protocol.

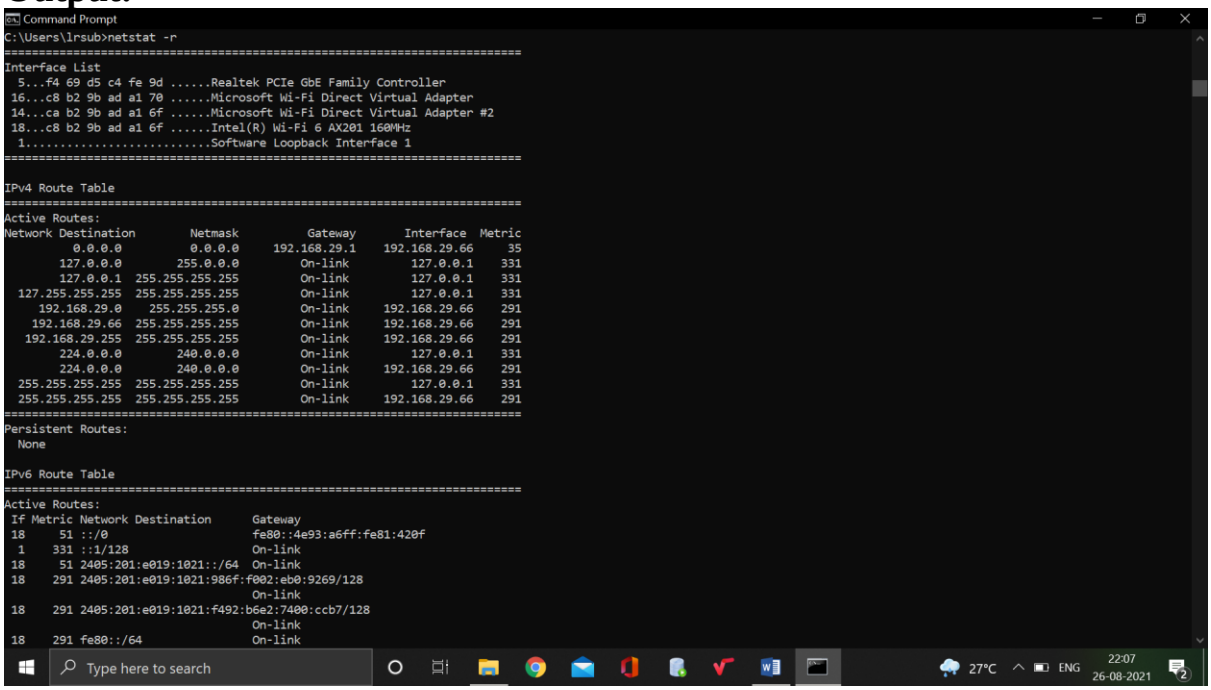
Output:

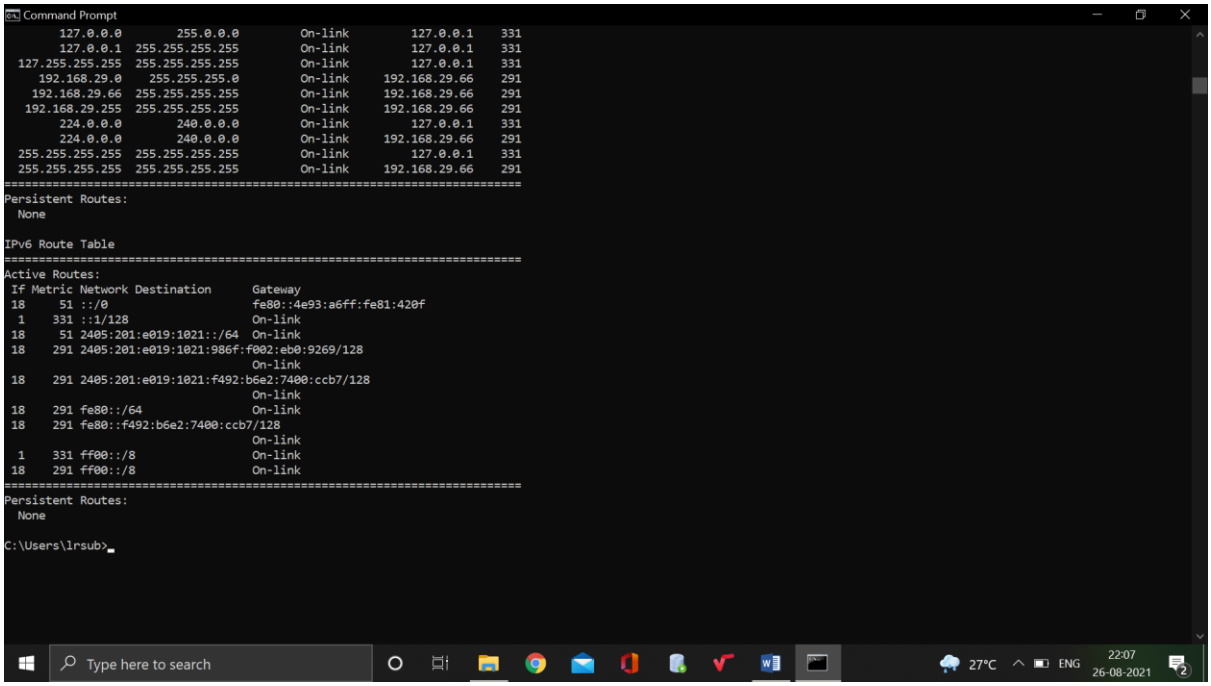


(h) netstat -r command:

- Execute netstat with **-r** to show the IP routing table.
- This is the same as using the route command to execute route print.

Output:

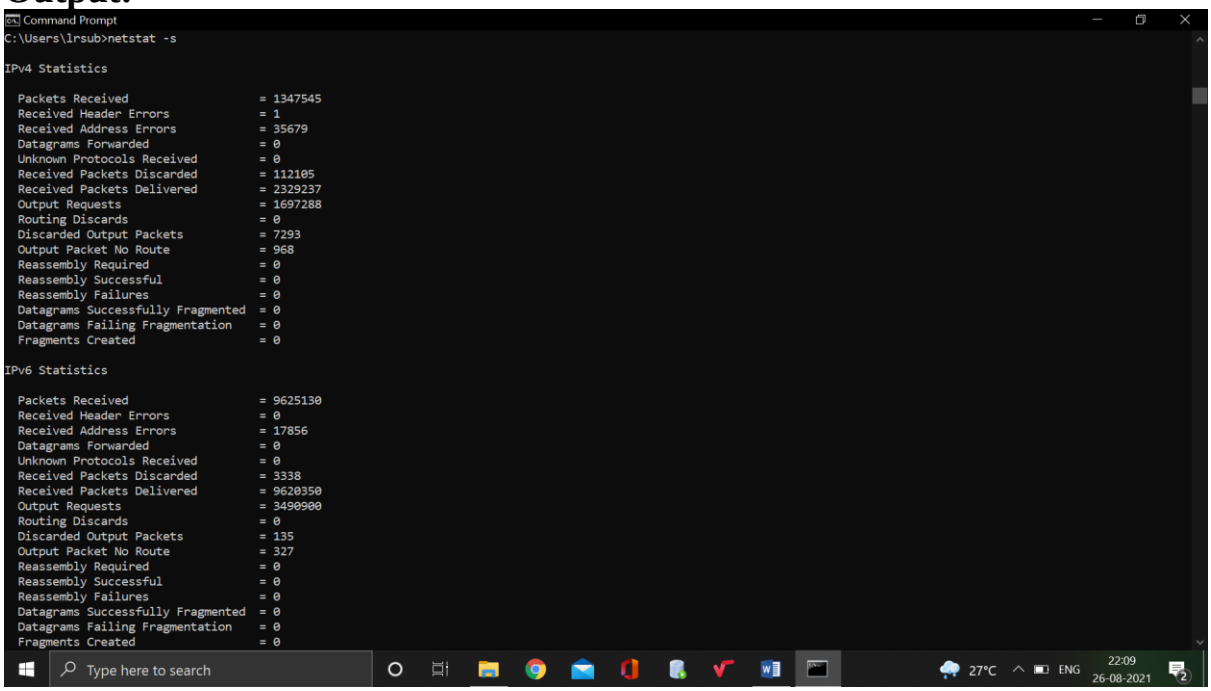


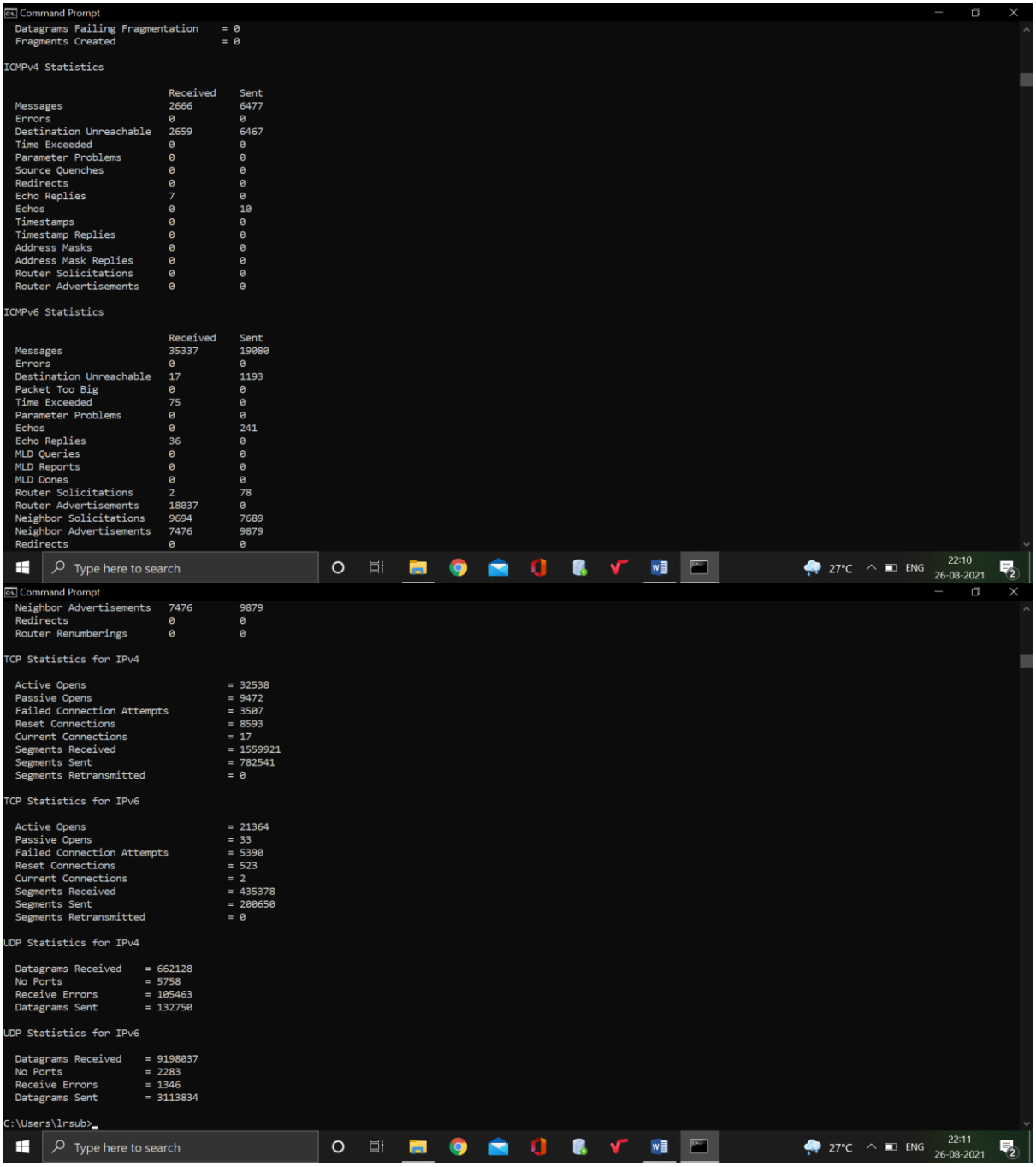


(i) netstat -s command:

- The **-s** option can be used with the netstat command to show detailed statistics by protocol.
- You can limit the statistics shown to a particular protocol by using the **-s** option and specifying that protocol, but be sure to use **-s** before **-p** protocol when using the switches together.

Output:

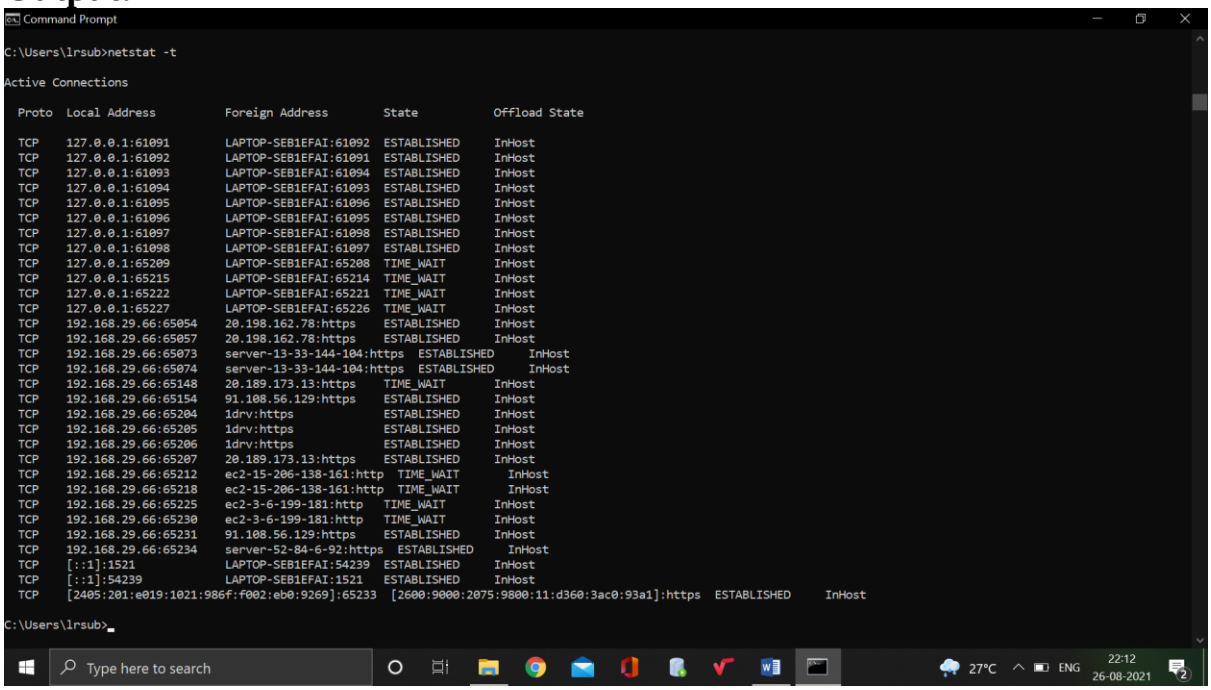




(j) netstat -t command:

- Use the **-t** switch to show the current TCP chimney offload state in place of the typically displayed TCP state.

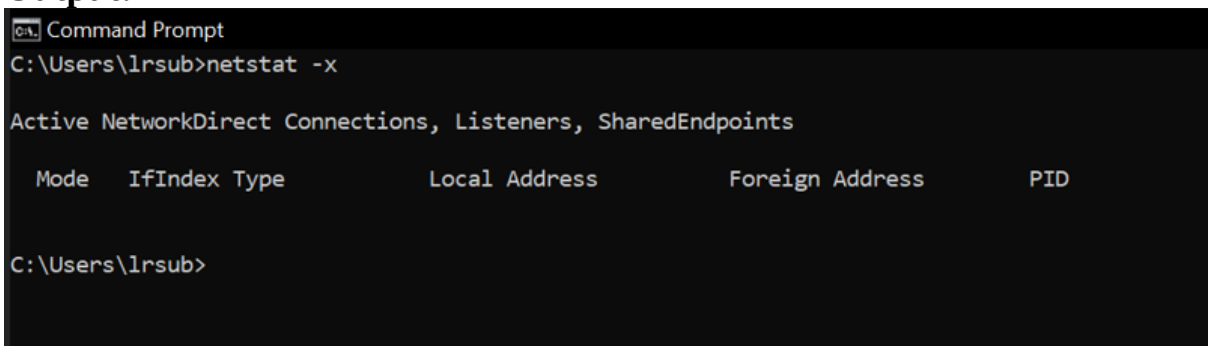
Output:



(k) netstat -x command:

- Use the **-x** option to show all NetworkDirect listeners, connections, and shared endpoints.

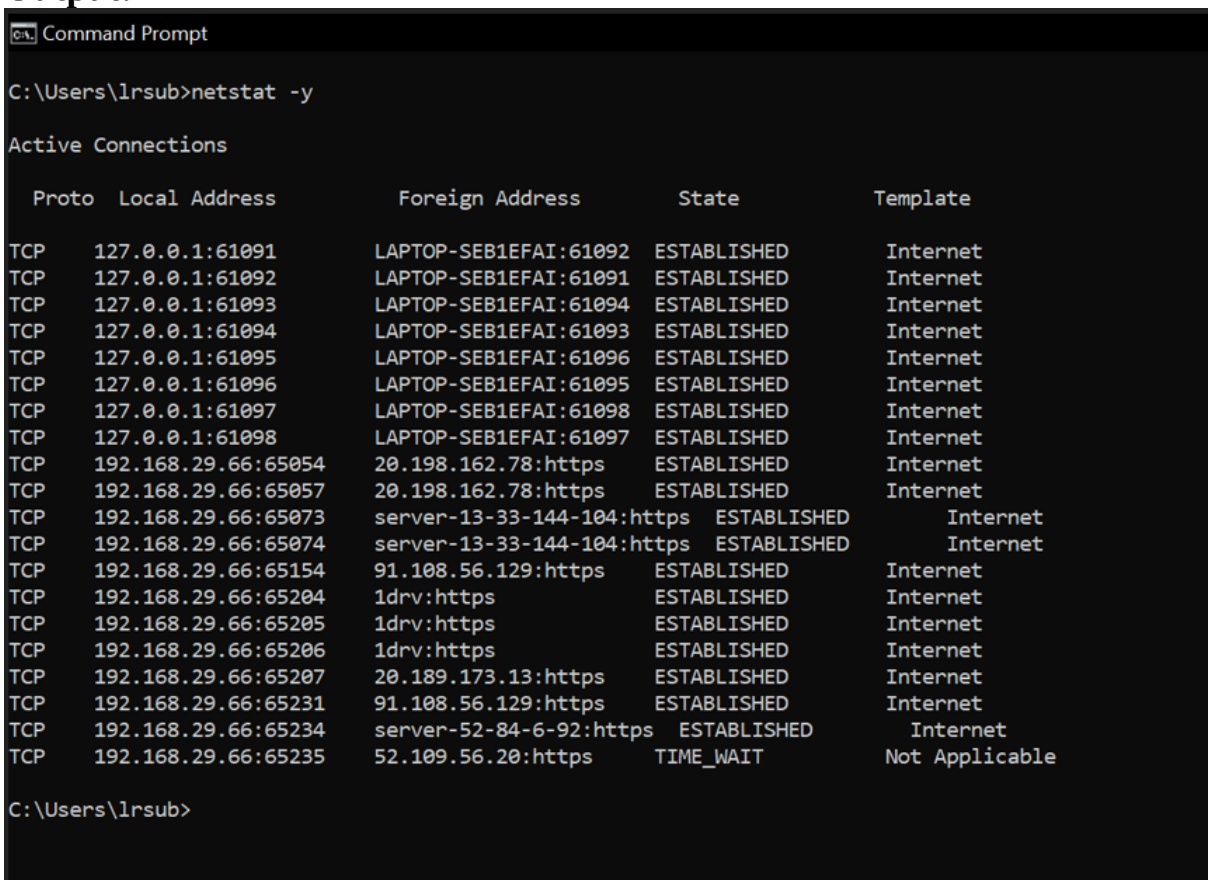
Output:



(l) netstat -y command:

- The **-y** switch can be used to show the TCP connection template for all connection. You cannot use **-y** with any other netstat option.

Output:

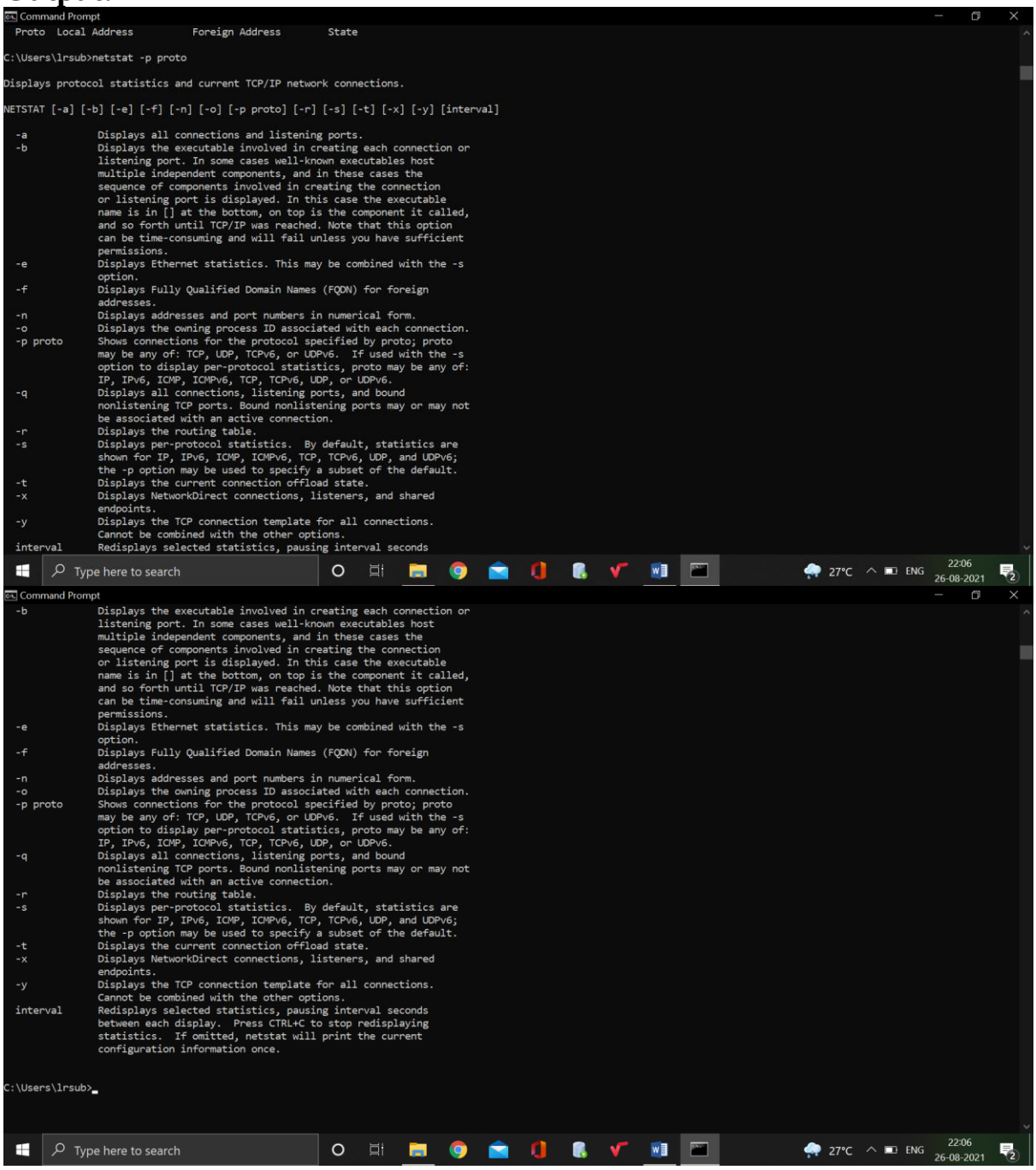


```
Command Prompt

C:\Users\lrsub>netstat -y

Active Connections

  Proto  Local Address          Foreign Address        State           Template

TCP    127.0.0.1:61091        LAPTOP-SEB1EFAI:61092  ESTABLISHED        Internet
TCP    127.0.0.1:61092        LAPTOP-SEB1EFAI:61091  ESTABLISHED        Internet
TCP    127.0.0.1:61093        LAPTOP-SEB1EFAI:61094  ESTABLISHED        Internet
TCP    127.0.0.1:61094        LAPTOP-SEB1EFAI:61093  ESTABLISHED        Internet
TCP    127.0.0.1:61095        LAPTOP-SEB1EFAI:61096  ESTABLISHED        Internet
TCP    127.0.0.1:61096        LAPTOP-SEB1EFAI:61095  ESTABLISHED        Internet
TCP    127.0.0.1:61097        LAPTOP-SEB1EFAI:61098  ESTABLISHED        Internet
TCP    127.0.0.1:61098        LAPTOP-SEB1EFAI:61097  ESTABLISHED        Internet
TCP    192.168.29.66:65054    20.198.162.78:https    ESTABLISHED        Internet
TCP    192.168.29.66:65057    20.198.162.78:https    ESTABLISHED        Internet
TCP    192.168.29.66:65073    server-13-33-144-104:https  ESTABLISHED       Internet
TCP    192.168.29.66:65074    server-13-33-144-104:https  ESTABLISHED       Internet
TCP    192.168.29.66:65154    91.108.56.129:https    ESTABLISHED        Internet
TCP    192.168.29.66:65204    1drv:https             ESTABLISHED        Internet
TCP    192.168.29.66:65205    1drv:https             ESTABLISHED        Internet
TCP    192.168.29.66:65206    1drv:https             ESTABLISHED        Internet
TCP    192.168.29.66:65207    20.189.173.13:https    ESTABLISHED        Internet
TCP    192.168.29.66:65231    91.108.56.129:https    ESTABLISHED        Internet
TCP    192.168.29.66:65234    server-52-84-6-92:https  ESTABLISHED        Internet
TCP    192.168.29.66:65235    52.109.56.20:https     TIME_WAIT          Not Applicable

C:\Users\lrsub>
```

(m) netstat -p proto command:

- When specifying a *protocol* with the **-p** option, you can use **tcp**, **udp**, **tcpv6**, or **udpv6**. If you use **-s** with **-p** to view statistics by protocol, you can use **icmp**, **ip**, **icmpv6**, or **ipv6** in addition to the first four I mentioned.

Output:



(n) netstat interval command:

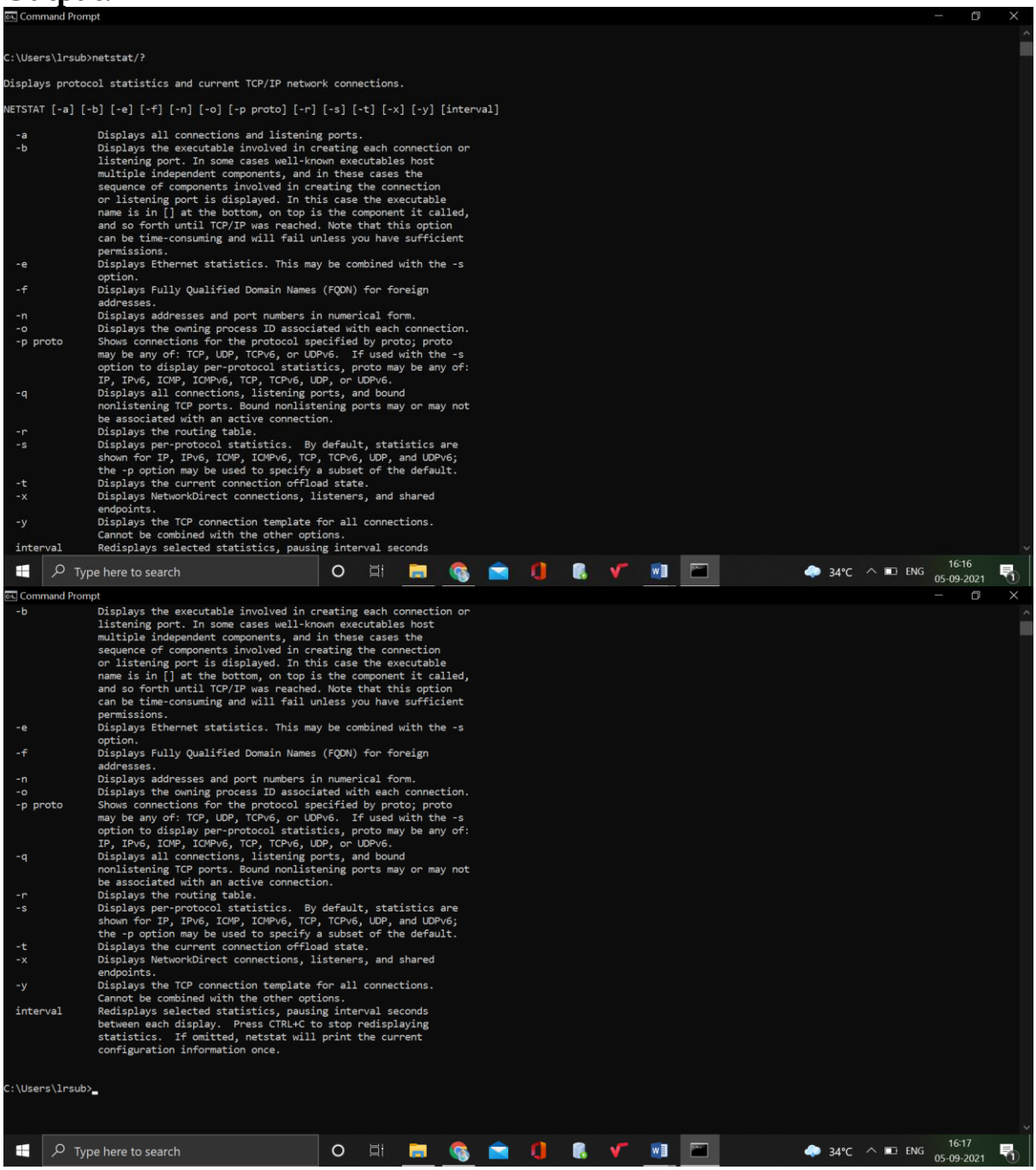
- This is the time, in seconds, that you'd like the netstat command to re-execute automatically, stopping only when you use **Ctrl-C** to end the loop.

Output:

(m) netstat/? command:

- Use the **help switch** to show details about the netstat command's several options.

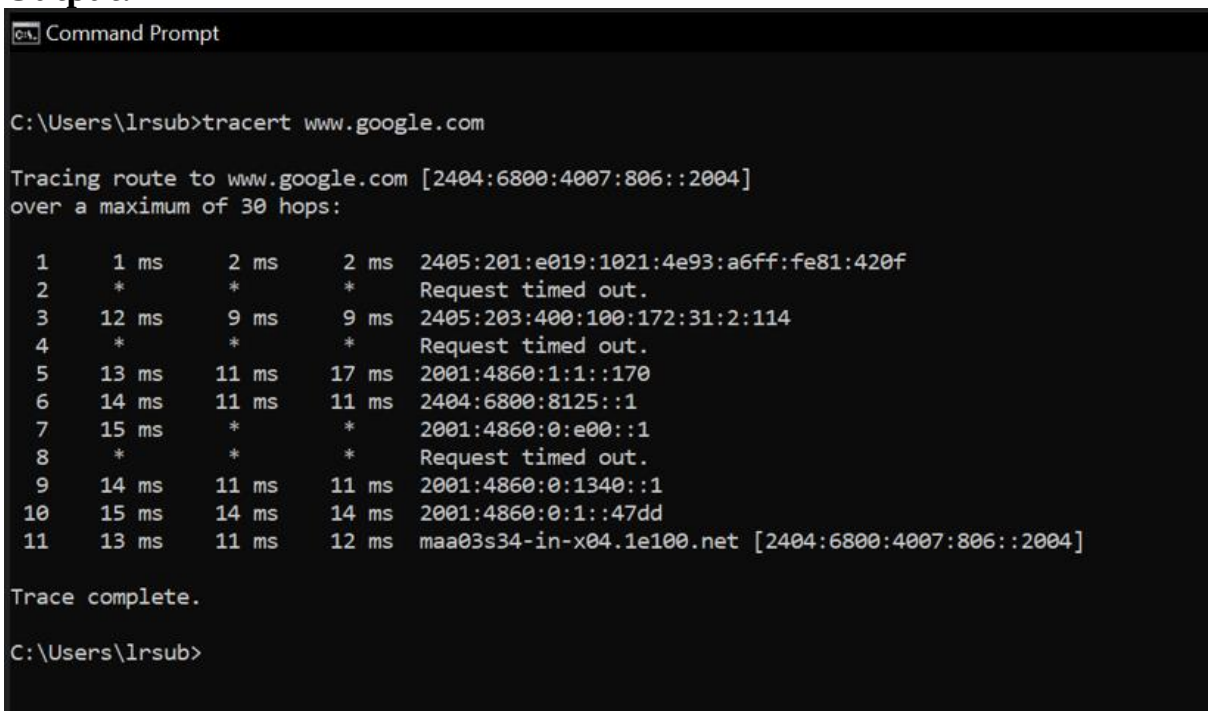
Output:



5. tracert command:

- TRACERT (Trace Route), is a command-line utility that you can use to trace the path that an Internet Protocol (IP) packet takes to its destination.
- Traceroute is a network diagnostic tool used to track in real-time the pathway taken by a packet on an IP network from source to destination, reporting the IP addresses of all the routers it pinged in between.
- Traceroute also records the time taken for each hop the packet makes during its route to the destination.

Output:



Result:

The basic commands are implemented successfully.

EX.NO. 2	APPLICATIONS USING TCP SOCKETS

EX.NO. 2A DATE: 09/09/2021	ECHO CLIENT & ECHO SERVER USING TCP SOCKETS

Aim:

To implement the java program for echo client and echo server using TCP sockets.

Algorithm:

Client side:

1. Start the program.
2. Create a socket which binds the IP address of server and the port address to acquire service.
3. after establishing connection send a data to server.
4. Receive and print the same data from server.
5. Close the socket.
6. End the program.

Server side:

1. Start the program.
2. Create a server socket to activate the port address.
3. Create a socket for the server socket which accepts the connection.
4. after establishing connection receive the data from client.
5. Print and send the same data to client.
6. Close the socket.
7. End the program.

Program:

Server side:

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

public class EchoServer {
   private Socket socket=null;
   private ServerSocket server=null;
   private DataInputStream in=null;
   public EchoServer(int port){
      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: ");
            in=new DataInputStream(new
BufferedInputStream(socket.getInputStream()));
            String line="";
            while(!line.equals("Over"))
            {
               try{
                  line=in.readUTF();
                  System.out.println(line);
               }
               catch(IOException e){
                  System.out.println(e);
               }
            }
            System.out.println("Closing connection.");
            socket.close();
            in.close();
         }
         catch(IOException i)
         {
            System.out.println(i);
         }
      }
      public static void main(String args[])
      {
         EchoServer server=new EchoServer(5000);
      }
}
```

Client side:

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

public class EchoClient {
   private Socket socket=null;
   private DataInputStream input=null;
   private DataOutputStream out=null;
   public EchoClient(String address,int port)
   {
      try
      {
```

```
            socket=new Socket(address,port);
            System.out.println("Connected with server");
            input=new DataInputStream(System.in);
            out=new DataOutputStream(socket.getOutputStream());
        }
        catch(UnknownHostException u)
        {
            System.out.println(u);
        }
        catch(IOException i)
        {
            System.out.println(i);
        }
        String line="";
        while(!line.equals("Over"))
        {
            try
            {
                line=input.readLine();
                out.writeUTF(line);
            }
            catch(IOException i)
            {
                System.out.println(i);
            }
        }
        try
        {
            input.close();
            out.close();
            socket.close();
        }
        catch(IOException i)
        {
            System.out.println(i);
        }
    }
    public static void main(String args[])
    {
        EchoClient client=new EchoClient("127.0.0.1",5000);
    }
}
```

Output:

Server:

```
Server started.
Waiting for a client.....
Client accepted:
Hi server
I hope the connection is successfully done
Bye server
Over
Closing connection.
BUILD SUCCESSFUL (total time: 32 seconds)
```

Client:

```
run:
Connected with server
Hello Server
I hope the connection is successfully done!
Bye server
Over
BUILD SUCCESSFUL (total time: 31 seconds)
```

Result:

Thus the java program for echo client and echo server using TCP sockets are written and executed successfully.

EX.NO. 2B DATE: 09/09/2021	CHAT APPLICATION USING TCP SOCKETS

Aim:

To write a client-server application for chat using TCP sockets.

Algorithm:

Client side:

1. Start the program.
2. Include necessary package in java.
3. To create a socket in client to server.
4. The client establishes a connection to the server.
5. The client accepts the connection and to send the data from client to server.
6. The client communicates the server to send the end of the message.
7. Stop the program.

Server side:

1. Start the program.
2. Include necessary package in java.
3. To create a socket in server to client.
4. The server establishes a connection to the client.
5. The server accept the connection and to send the data from server to client and vice versa.
6. The server communicates the client to send the end of the message.
7. Stop the program.

Program:

Server side:

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

public class ChatServer {
   public static void main(String[] args)throws Exception
   {
      ServerSocket ss=new ServerSocket(2000);
      Socket sk=ss.accept();
      BufferedReader cin=new BufferedReader(new
InputStreamReader(sk.getInputStream()));
      PrintStream cout=new PrintStream(sk.getOutputStream());
      BufferedReader stdin=new BufferedReader(new
InputStreamReader(System.in));
```

```
        String s;
        while(true)
        {
            s=cin.readLine();
            if(s.equalsIgnoreCase("End"))
            {
                cout.println("Bye");
                break;
            }
            System.out.println("Client: "+s);
            System.out.print("Server: ");
            s=stdin.readLine();
            cout.println(s);
        }
        ss.close();
        sk.close();
        cin.close();
        cout.close();
    }
}
```

Client side:

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

public class ChatClient {
    public static void main(String[] args)throws Exception
    {
        Socket sk=new Socket("127.0.0.1",2000);
        BufferedReader sin=new BufferedReader(new
InputStreamReader(sk.getInputStream()));
        PrintStream sout=new PrintStream(sk.getOutputStream());
        BufferedReader stdin=new BufferedReader(new
InputStreamReader(System.in));
        String s;
        while(true)
        {
            System.out.print("Client: ");
            s=stdin.readLine();
            sout.println(s);
            s=sin.readLine();
            System.out.println("Server: "+s);
```

```
            if(s.equalsIgnoreCase("Bye"))
                break;
        }
        sk.close();
        sin.close();
        sout.close();
        stdin.close();
    }
}
```

Output:

Server:

```
Client: Hi server
Server: Hi client. How can I help you?
Client: I am just checking if the connection is successful.
Server: Yes, of course. It is successfully connected.
Client: That's great. Thank you server. Bye
Server: It's my pleasure client. Bye and take care.
End
```

Client:

```
Client: Hi server
Server: Hi client. How can I help you?
Client: I am just checking if the connection is successful.
Server: Yes, of course. It is successfully connected.
Client: That's great. Thank you server. Bye
Server: It's my pleasure client. Bye and take care.
Client: Bye
```

Result:

Thus the java program for chat application between server and client using TCP sockets are written and executed successfully.

EX.NO. 2C DATE: 09/09/2021	FILE TRANSFER BETWEEN CLIENT & SERVER USING TCP SOCKETS

Aim:

To perform file transfer between client and server using TCP sockets.

Algorithm:

Client side:

1. Start.
2. Establish a connection between the Client and Server.
3. Sockets s = new Socket(InetAddress. getLocalHost(), 1100);
4. Implement a client that can send two requests.
    i) To get a file from the server.
    ii) To put or send a file to the server.
5. After getting approval from the server, the clients either get file from the server or send file to the server.

Server side:

1. Start.
2. Implement a server socket that listens to a particular port number.
3. Server reads the filename and sends the data stored in the file for the‘get’ request.
4. It reads the data from the input stream and writes it to a file in the server for the ‘put’ instruction.
5. Exit upon client’s request.
6. Stop.

Program:

Server side:

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

public class fileserver {
   public static void main(String[] args)throws Exception{
      ServerSocket ssock=new ServerSocket(5000);
      Socket socket=ssock.accept();
      InetAddress ia=InetAddress.getByName("localhost");
      File f=new File("D:\\Sonal College\\5th sem files\\Computer
Networks\\Lab\\fileTransfer.txt");
      FileInputStream fin=new FileInputStream(f);
      BufferedInputStream bis=new BufferedInputStream(fin);
      OutputStream os=socket.getOutputStream();
```

```
        byte[] contents;
        long fileLength=f.length();
        long current=0;
        long start=System.nanoTime();
        while(current!=fileLength)
        {
           int size=10000;
           if(fileLength-current>=size)
              current+=size;
           else
           {
              size=(int)(fileLength-current);
              current=fileLength;
           }
           contents=new byte[size];
           bis.read(contents,0,size);
           os.write(contents);
           System.out.println("Sending file....."+(current*100)/fileLength+"%
complete");
        }
        os.flush();
        socket.close();
        ssock.close();
        System.out.println("File sent successfully!");
    }
}
```

Client side:

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

public class fileclient {
    public static void main(String[] args)throws Exception{
        Socket socket=new Socket(InetAddress.getByName("localhost"),5000);
        byte[] contents=new byte[10000];
        FileOutputStream fout=new FileOutputStream("D:\\lrsub\\Sonal
College\\5th sem files\\Computer Networks\\Lab\\fileTransfer1.txt");
        BufferedOutputStream bos=new BufferedOutputStream(fout);
        InputStream is=socket.getInputStream();
        int bytesRead=0;
        while((bytesRead=is.read(contents))!=-1)
        {
```

```
            bos.write(contents,0,bytesRead);
        }
        bos.flush();
        socket.close();
        System.out.println("File saved successfully!");
    }
}
```

Output:

Server:

```
Sending file.....100% complete
File sent successfully!
BUILD SUCCESSFUL (total time: 11 seconds)
```

Client:

```
File saved successfully!
BUILD SUCCESSFUL (total time: 1 second)
```

Result:

Thus the java program for file transfer between client and server using TCP sockets are written and executed successfully.

EX.NO. 3 DATE: 09/09/2021	HTTP WEB CLIENT PROGRAM TO DOWNLOAD A WEB PAGE USING TCP SOCKETS

Aim:

To download a webpage using TCP sockets

Algorithm:

Client side:

1. Start the program.
2. Create a socket which binds the Ip address of server and the port address to acquire service.
3. After establishing connection send the url to server.
4. Open a file and store the received data into the file.
5. Close the socket.
6. End the program.

Server side:

1. Start the program.
2. Create a server socket to activate the port address.
3. Create a socket for the server socket which accepts the connection.
4. After establishing connection receive url from client.
5. Download the content of the url received and send the data to client.
6. Close the socket.
7. End the program.

Program:

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;

public class urldownload {
   private static final int BUFFER_SIZE = 4096;
   public static void downloadFile(String fileURL, String saveDir)
   throws IOException {
      URL url = new URL(fileURL);
      HttpURLConnection httpConn = (HttpURLConnection)
url.openConnection();
      int responseCode = httpConn.getResponseCode();
      if (responseCode == HttpURLConnection.HTTP_OK) {
```

```
            String fileName = "";
            String disposition = httpConn.getHeaderField("Content-Disposition");
            String contentType = httpConn.getContentType();
            int contentLength = httpConn.getContentLength();
            if (disposition != null) {
               int index = disposition.indexOf("filename=");
               if (index > 0) {
                  fileName = disposition.substring(index + 10,
                        disposition.length() - 1);
               }
            }
            else {
               fileName = fileURL.substring(fileURL.lastIndexOf("/") + 1,
                     fileURL.length());
            }
            System.out.println("Content-Type = " + contentType);
            System.out.println("Content-Disposition = " + disposition);
            System.out.println("Content-Length = " + contentLength);
            System.out.println("fileName = " + fileName);
            InputStream inputStream = httpConn.getInputStream();
            String saveFilePath = saveDir + File.separator + fileName;
            FileOutputStream outputStream = new FileOutputStream(saveFilePath);
            int bytesRead = -1;
            byte[] buffer = new byte[BUFFER_SIZE];
            while ((bytesRead = inputStream.read(buffer)) != -1) {
               outputStream.write(buffer, 0, bytesRead);
            }
            outputStream.close();
            inputStream.close();
            System.out.println("File downloaded");
         }
           else {
            System.out.println("No file to download. Server replied HTTP code: " +
responseCode);
         }
         httpConn.disconnect();
      }
      public static void main(String[] args) {
         String fileURL = "
https://www.abc.net.au/reslib/200709/r186930_1738931.jpg ";
         String saveDir = " D:\Sonal College\5th sem files\Computer Networks ";
         try {
            http.downloadFile(fileURL, saveDir);
```

```
        }
          catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

Output:

```
run:
Content-Type=image/jpeg
Content-Disposition=null
Content-Length=86851
Filename=r186930_1738931.jpg
File downloaded
BUILD SUCCESSFUL (total time: 0 seconds)
```

Downloaded file:

Result:

Thus the java program for HTTP web client to download a webpage using TCP sockets were written and successfully executed.

EX.NO. 3 DATE: 09/09/2021	HTTP WEB CLIENT PROGRAM TO DOWNLOAD A WEB PAGE USING TCP SOCKETS

Aim:

To download a webpage using TCP sockets

Algorithm:

Client side:

1. Start the program.
2. Create a socket which binds the Ip address of server and the port address to acquire service.
3. After establishing connection send the url to server.
4. Open a file and store the received data into the file.
5. Close the socket.
6. End the program.

Server side:

1. Start the program.
2. Create a server socket to activate the port address.
3. Create a socket for the server socket which accepts the connection.
4. After establishing connection receive url from client.
5. Download the content of the url received and send the data to client.
6. Close the socket.
7. End the program.

Program:

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;

public class urldownload {
   private static final int BUFFER_SIZE = 4096;
   public static void downloadFile(String fileURL, String saveDir)
   throws IOException {
      URL url = new URL(fileURL);
      HttpURLConnection httpConn = (HttpURLConnection)
url.openConnection();
      int responseCode = httpConn.getResponseCode();
      if (responseCode == HttpURLConnection.HTTP_OK) {
```

```
            String fileName = "";
            String disposition = httpConn.getHeaderField("Content-Disposition");
            String contentType = httpConn.getContentType();
            int contentLength = httpConn.getContentLength();
            if (disposition != null) {
               int index = disposition.indexOf("filename=");
               if (index > 0) {
                  fileName = disposition.substring(index + 10,
                        disposition.length() - 1);
               }
            }
            else {
               fileName = fileURL.substring(fileURL.lastIndexOf("/") + 1,
                     fileURL.length());
            }
            System.out.println("Content-Type = " + contentType);
            System.out.println("Content-Disposition = " + disposition);
            System.out.println("Content-Length = " + contentLength);
            System.out.println("fileName = " + fileName);
            InputStream inputStream = httpConn.getInputStream();
            String saveFilePath = saveDir + File.separator + fileName;
            FileOutputStream outputStream = new FileOutputStream(saveFilePath);
            int bytesRead = -1;
            byte[] buffer = new byte[BUFFER_SIZE];
            while ((bytesRead = inputStream.read(buffer)) != -1) {
               outputStream.write(buffer, 0, bytesRead);
            }
            outputStream.close();
            inputStream.close();
            System.out.println("File downloaded");
         }
           else {
            System.out.println("No file to download. Server replied HTTP code: " +
responseCode);
         }
         httpConn.disconnect();
      }
      public static void main(String[] args) {
         String fileURL = "
https://www.abc.net.au/reslib/200709/r186930_1738931.jpg ";
         String saveDir = " D:\Sonal College\5th sem files\Computer Networks ";
         try {
            http.downloadFile(fileURL, saveDir);
```

```
        }
          catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

Output:

```
run:
Content-Type=image/jpeg
Content-Disposition=null
Content-Length=86851
Filename=r186930_1738931.jpg
File downloaded
BUILD SUCCESSFUL (total time: 0 seconds)
```

Downloaded file:

Result:

Thus the java program for HTTP web client to download a webpage using TCP sockets were written and successfully executed.

EX.NO. 5 DATE: 20/10/2021	SIMULATION OF ERROR DETECTION USING CRC

Aim:

To write a java program for the simulation of error detection using CRC.

Algorithm:

1. Open the editor and type the program for error detection
2. Get the input in the form of bits.
3. Append the redundancy bits.
4. Divide the appended data using a divisor polynomial.
5. The resulting data should be transmitted to the receiver.
6. At the receiver the received data is entered.
7. The same process is repeated at the receiver.
8. If the remainder is zero there is no error otherwise there is some error in the received bits.
9. Run the program.

Program:

```
import java.io.*;
public static void main(String args[]) throws IOException
{
 BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
 System.out.println("Enter Generator:");
 String gen = br.readLine();
 System.out.println("Enter Data:");
 String data = br.readLine();
 String code = data;
 while(code.length() < (data.length() + gen.length() - 1))
  code = code + "0";
  code = data + div(code,gen);
 System.out.println("The transmitted Code Word is: " + code);
 System.out.println("Please enter the received Code Word: ");
 String rec = br.readLine();
 if(Integer.parseInt(div(rec,gen)) == 0)
  System.out.println("The received code word contains no errors.");
 else
  System.out.println("The received code word contains errors.");
}

static String div(String num1,String num2)
{
 int pointer = num2.length();
 String result = num1.substring(0, pointer);
 String remainder = "";
 for(int i = 0; i < num2.length(); i++)
 {
  if(result.charAt(i) == num2.charAt(i))
   remainder += "0";
  else
   remainder += "1";
```

```
 }
 while(pointer < num1.length())
 {
  if(remainder.charAt(0) == '0')
  {
   remainder = remainder.substring(1, remainder.length());
   remainder = remainder + String.valueOf(num1.charAt(pointer));
   pointer++;
  }
  result = remainder;
  remainder = "";
  for(int i = 0; i < num2.length(); i++)
  {
   if(result.charAt(i) == num2.charAt(i))
    remainder += "0";
   else
    remainder += "1";
  }
 }
 return remainder.substring(1,remainder.length());
}
```

Output:

```
run:
Enter Generator:
1101
Enter Data:
100100
The transmitted Code Word is: 100100001
Please enter the received Code Word:
100100001
The received code word contains no errors.
BUILD SUCCESSFUL (total time: 33 seconds)
```

Result:

Thus the java program for simulation of error detection using CRC were written and successfully executed.

EX.NO. 6 DATE: 20/10/2021	IMPLEMENTATION OF DISTANCE VECTOR ROUTING AND LINK STATE ROUTING ALGORITHM

Aim:

To simulate and observe traffic route of a network using distance vector and link state routing algorithm.

Algorithm:

1. Create a simulator object.
2. Set routing protocol to distance vector routing.
3. Track packets on all links on to NAM trace and text trace file.
4. Define finish procedure to close files, flash tracing and run NAM.
5. Create five nodes.
6. Specify the link characteristics between the nodes.
7. Describe their layout topology as an octagon.
8. Add UDP agent for node n_o.
9. Create CBR traffic on the top of the UDP and set the traffic traffic parameters.
10. Add NULL agent to node n_3.
11. Connect source and sink.
12. Schedule as follows:
    - Start traffic flow at 1.0
    - Down the link n_1-n_2 at 15.0
    - Up the link n_1-n_2 at 25.0
    - Call finish procedure at 35.0
13. Start the schedule.
14. Observe the traffic route when the link is up and down.
15. View the simulated events and trace file, analyze it.
16. Stop.

6A. Distance Vector Routing:

Program:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Distance_Vectore_Routing {
   static int graph[][];
   static int via[][];
   static int rt[][];
   static int v;
   static int e;
public static void main(String args[]) throws IOException{
  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
  System.out.println("Enter the number of Vertices: ");
  v = Integer.parseInt(br.readLine());
  System.out.println("Enter the number of Edges: ");
  e = Integer.parseInt(br.readLine());
  graph = new int[v][v];
  via = new int[v][v];
  rt = new int[v][v];
```

```
  for(int i = 0; i < v; i++)
  for(int j = 0; j < v; j++)
  {
    if(i == j)
      graph[i][j] = 0;
    else
      graph[i][j] = 9999;
  }
  for(int i = 0; i < e; i++)
 {
    System.out.println("Enter data for Edge " + (i + 1) + ":");
    System.out.print("Source: ");
    int s = Integer.parseInt(br.readLine());
    s--;
    System.out.print("Destination: ");
    int d = Integer.parseInt(br.readLine());
    d--;
    System.out.print("Cost: ");
    int c = Integer.parseInt(br.readLine());
    graph[s][d] = c;
    graph[d][s] = c;
  }
  dvr_calc_disp("The initial Routing Tables are: ");
  System.out.print("Please enter the Source Node for the edge whose cost has changed: ");
  int s = Integer.parseInt(br.readLine());
  s--;
  System.out.print("Please enter the Destination Node for the edge whose cost has changed:
");
  int d = Integer.parseInt(br.readLine());
  d--;
  System.out.print("Please enter the new cost: ");
  int c = Integer.parseInt(br.readLine());
  graph[s][d] = c;
  graph[d][s] = c;
 dvr_calc_disp("The new Routing Tables are: ");
 }
 static void dvr_calc_disp(String message)
{
  System.out.println();
  init_tables();
  update_tables();
  System.out.println(message);
  print_tables();
  System.out.println();
}

static void update_table(int source)
{
  for(int i = 0; i < v; i++)
  {
```

```
   if(graph[source][i] != 9999)
  {
     int dist = graph[source][i];
     for(int j = 0; j < v; j++)
     {
        int inter_dist = rt[i][j];
        if(via[i][j] == source)
          inter_dist = 9999;
        if(dist + inter_dist < rt[source][j])
        {
          rt[source][j] = dist + inter_dist;
          via[source][j] = i;
        }
     }
  }
 }
}

static void update_tables()
{
  int k = 0;
  for(int i = 0; i < 4*v; i++)
  {
     update_table(k);
     k++;
     if(k == v)
     k = 0;
  }
 }

static void init_tables()
{
   for(int i = 0; i < v; i++)
  {
     for(int j = 0; j < v; j++)
     {
        if(i == j)
        {
          rt[i][j] = 0;
          via[i][j] = i;
        }
       else
       {
         rt[i][j] = 9999;
         via[i][j] = 100;
        }
     }
   }
 }
```

```
static void print_tables()
{
   for(int i = 0; i < v; i++)
   {
      for(int j = 0; j < v; j++)
     {
         System.out.print("Dist: " + rt[i][j] + "    ");
      }
      System.out.println();
    }
  }
}
```

Output:

```
run:
Enter the number of Vertices:
4
Enter the number of Edges:
5
Please enter data for Edge 1:
Source: 1
Destination: 3
Cost: 3
Please enter data for Edge 2:
Source: 1
Destination: 2
Cost: 4
Please enter data for Edge 3:
Source: 1
Destination: 4
Cost: 3
Please enter data for Edge 4:
Source: 2
Destination: 4
Cost: 3
Please enter data for Edge 5:
Source: 2
Destination: 3
Cost: 5
The initial Routing Tables are:
Dist: 0    Dist: 4    Dist: 3    Dist: 3
Dist: 4    Dist: 0    Dist: 5    Dist: 3
Dist: 3    Dist: 5    Dist: 0    Dist: 6
Dist: 3    Dist: 3    Dist: 6    Dist: 0

Please enter the Source Node for the edge whose cost has changed: 2
Please enter the Destination Node for the edge whose cost has changed: 3
Please enter the new cost: 1

The new Routing Tables are:
Dist: 0    Dist: 4    Dist: 3    Dist: 3
Dist: 4    Dist: 0    Dist: 1    Dist: 3
Dist: 3    Dist: 1    Dist: 0    Dist: 4
Dist: 3    Dist: 3    Dist: 4    Dist: 0

BUILD SUCCESSFUL (total time: 1 minute 10 seconds)
```

6B. Link State Routing:

Program:

```
import java.util.Scanner;
public class Link_State_Routing {
    public  int distance[] = new int[10];
    public  int cost[][] = new int[10][10];
    public void calc(int n,int s)
   {
      int flag[] = new int[n+1];
      int i,minpos=1,k,c,minimum;
      for(i=1;i<=n;i++)
     {
         flag[i]=0;
         this.distance[i]=this.cost[s][i];
     }
      c=2;
      while(c<=n)
     {
        minimum=99;
        for(k=1;k<=n;k++)
       {
           if(this.distance[k]<minimum && flag[k]!=1)
          {
            minimum=this.distance[i];
            minpos=k;
          }
       }
        flag[minpos]=1;
        c++;
        for(k=1;k<=n;k++)
       {
           if(this.distance[minpos]+this.cost[minpos][k] <  this.distance[k] && flag[k]!=1 )
             this.distance[k]=this.distance[minpos]+this.cost[minpos][k];
       }
     }
  }
  public static void main(String args[])
 {
    int nodes,source,i,j;
    Scanner in = new Scanner(System.in);
    System.out.println("Enter the Number of Nodes \n");
    nodes = in.nextInt();
    Link_State_Routing d = new Ex6A_Link_State_Routing();
    System.out.println("Enter the Cost Matrix Weights: \n");
    for(i=1;i<=nodes;i++)
    for(j=1;j<=nodes;j++)
    {
      d.cost[i][j]=in.nextInt();
      if(d.cost[i][j]==0)
      d.cost[i][j]=999;
```

```
    }
    System.out.println("Enter the Source Vertex :\n");
    source=in.nextInt();
    d.calc(nodes,source);
    System.out.println("The Shortest Path from Source \t"+source+"\t to all other vertices are :
\n");
    for(i=1;i<=nodes;i++)
      if(i!=source)
        System.out.println("source :"+source+"\t destination :"+i+"\t MinCost is
:"+d.distance[i]+"\t");
   }
}
```

Output:

```
run:
Enter the Number of Nodes

5
Enter the Cost Matrix Weights:

0 1 0 2 0
1 0 1 0 4
0 1 0 0 3
2 0 0 0 3
0 4 3 3 0
Enter the Source Vertex :

1
The Shortest Path from Source   1       to all other vertices are :

source :1       destination :2  MinCost is :1
source :1       destination :3  MinCost is :2
source :1       destination :4  MinCost is :2
source :1       destination :5  MinCost is :5
BUILD SUCCESSFUL (total time: 53 seconds)
```

Observation	
Record	
Total	
Staff Signature	

Result:

Thus the java program to simulate and observe traffic route of a network using distance vector and link state routing algorithm were written and successfully executed.

EX.NO. 7 DATE: 28/10/2021	SIMULATION OF DNS USING UDP SOCKETS

Aim:

To write a java program for the simulation of DNS using UDP sockets.

Algorithm:

1. Start the program.
2. Get the frame size from the user.
3. To create the frame based on the user request.
4. To send frames to server from the client side.
5. If your frames reach the server, it will send ACK signal to client otherwise it will send NACK signal to the client.
6. Stop the program.

Program:

UDP DNS Server:

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

public class UDP_DNS_Server {
   private static int indexOf(String[] array, String str){
      str = str.trim();
      for (int i=0; i < array.length; i++)
      {
         if (array[i].equals(str)) return i;
      }
      return -1;
   }
   public static void main(String arg[])throws IOException{
      String[] hosts = {"zoho.com", "gmail.com","google.com", "facebook.com"};
      String[] ip = {"172.28.251.59", "172.217.11.5","172.217.11.14","31.13.71.36"};
      System.out.println("Press Ctrl + C to Quit");
      while (true){
         DatagramSocket serversocket=new DatagramSocket(1362);
         byte[] senddata = new byte[1021];
         byte[] receivedata = new byte[1021];
         DatagramPacket recvpack = new DatagramPacket(receivedata,
         receivedata.length);
         serversocket.receive(recvpack);
         String sen = new String(recvpack.getData());
         InetAddress ipaddress = recvpack.getAddress();
         int port = recvpack.getPort();
         String capsent;
         System.out.println("Request for host " + sen);
         if(indexOf (hosts, sen) != -1)
            capsent = ip[indexOf (hosts, sen)];
         else
```

```
            capsent = "Host Not Found"; senddata = capsent.getBytes();
          DatagramPacket pack = new DatagramPacket (senddata,
          senddata.length,ipaddress,port);
          serversocket.send(pack);
          serversocket.close();
       }
    }
}
```

UDP DNS Client:

```
public class UDP_DNS_Client {
   public static void main(String args[])throws IOException{
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      DatagramSocket clientsocket = new DatagramSocket();
      InetAddress ipaddress;
      if (args.length == 0)
         ipaddress = InetAddress.getLocalHost();
      else
         ipaddress = InetAddress.getByName(args[0]);
      byte[] senddata = new byte[1024];
      byte[] receivedata = new byte[1024];
      int portaddr = 1362;
      System.out.print("Enter the hostname : ");
      String sentence = br.readLine();
      senddata = sentence.getBytes();
      DatagramPacket pack = new
DatagramPacket(senddata,senddata.length,ipaddress,portaddr);
      clientsocket.send(pack);
      DatagramPacket recvpack =new DatagramPacket(receivedata,receivedata.length);
      clientsocket.receive(recvpack);
      String modified = new String(recvpack.getData());
      System.out.println("IP Address: " + modified);
      clientsocket.close();
   }
}
```

Output:

Server side:

```
run:
Press Ctrl + C to Quit
Request for host google.com
```

Client side:

```
run:
Enter the hostname : google.com
IP Address: 172.217.11.14
BUILD SUCCESSFUL (total time: 13 seconds)
```

Observation	
Record	
Total	
Staff Signature	

Result:

Thus the java program for the simulation of DNS using UDP sockets were written and successfully executed.

EX.NO. 8 DATE: 12/11/2021	SWITCH CONFIGURATION USING PACKET TRACER

Aim:

To establish a basic switch configuration between the end devices.

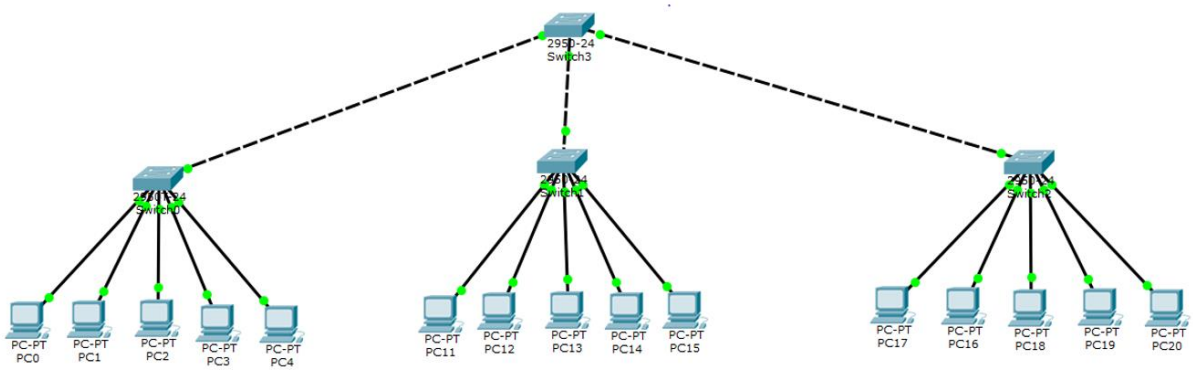
Algorithm:

1. Configure the end devices.
2. Establish connection from source to destination through switches.
3. Enable the switch.
4. Configure console password and vty password.

Commands:

```
Switch>enable
Switch#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Switch(config)#hostname cse
cse(config)#
cse (config)#line console 0
cse(config-line)#password lab1
cse (config-line)#login
cse (config-line)#line vty 0 15
cse (config-line)#password lab1
cse (config-line)#login
cse (config-line)#exit
```

Output:



Observation	
Record	
Total	
Staff Signature	

Result:

Thus the switch configuration using packet tracer is performed and verified.

EX.NO. 9 DATE: 12/11/2021	ROUTER CONFIGURATION USING PACKET TRACER

Aim:

To establish a basic router configuration between the end devices.

Algorithm:

1. Configure the end devices.
2. Establish connection from source to destination through switches.
3. Click the router and enter the code configuration in cli tab.

Commands:

ROUTER 1:

```
Router>enable
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface serial 2/0
Router(config-if)#ip address 172.16.1.1 255.255.0.0
Router(config-if)#no shutdown

Router(config-if)#
%LINK-5-CHANGED: Interface Serial 2/0, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface serial0/0, changed state to up
Router(config-if)#interface serial 2/0
Router(config-if)#ip address 172.16.1.3 255.255.0.0
Router(config-if)#no shutdown

Router(config-if)#
%LINK-5-CHANGED: Interface serial 2/0, changed state to up

Router#
%SYS-5-CONFIG_I: Configured from console by console
```

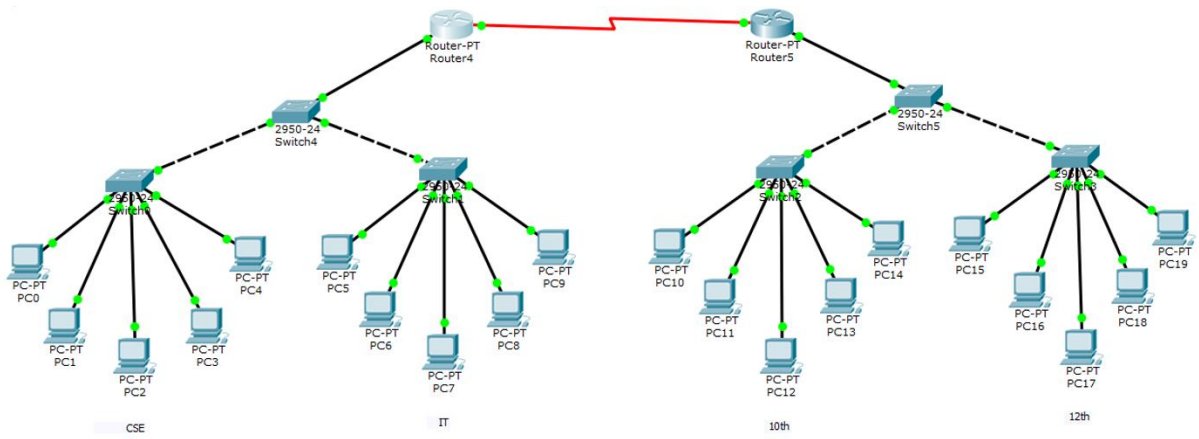
ROUTER 2:

```
Router>enable
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface serial 2/0
Router(config-if)#ip address 172.16.3.4 255.255.0.0
Router(config-if)#no shutdown

Router(config-if)#
%LINK-5-CHANGED: Interface serial 2/0, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface serial 2/0, changed state to up
Router(config)#interface serial 2/0
Router(config-if)#ip address 172.16.3.4  255.255.255.0
Router(config-if)#no shutdown
```

Router(config-if)#
%LINK-5-CHANGED: Interface serial 2/0, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface serial 2/0, changed state to up

Output:



Observation	
Record	
Total	
Staff Signature	

Result:

Thus the router configuration using packet tracer is performed and verified.

EX.NO. 10A DATE: 12/11/2021	OSPF CONFIGURATION USING PACKET TRACER

Aim:

To establish the OSPF (OPEN SHORTEST PATH FIRST) configuration between the end devices.

Algorithm:

1. Configure the end devices.
2. Configure connection from source to destination through routers
3. Enable the router configuration
4. Find the best path from source to destination.

Commands:

ROUTER 1:

```
R1(config)#int fa 0/0
R1(config-if)#ip add 10.0.0.1 255.0.0.0
R1(config-if)#no shut
R1(config-if)#
R1(config-if)#int serial 0/0/0
R1(config-if)#ip add 20.0.0.1 255.0.0.0
R1(config-if)#no shut
```

ROUTER 2:

```
R2(config-if)#int fa0/0
R2(config-if)#ip add 30.0.0.1 255.0.0.0
R2(config-if)#no shut
R2(config-if)#
R2(config-if)#int serial0/0/0
R2(config-if)#ip address 20.0.0.2 255.0.0.0
R2(config-if)#no shut
```

PC1 IP add 10.0.0.2 Subnet mask 255.0.0.0 Default gateway 10.0.0.1

PC2 IP add 30.0.0.2 Subnet mask 255.0.0.0 Default gateway 30.0.0.1

Configure OSPF on the routers.

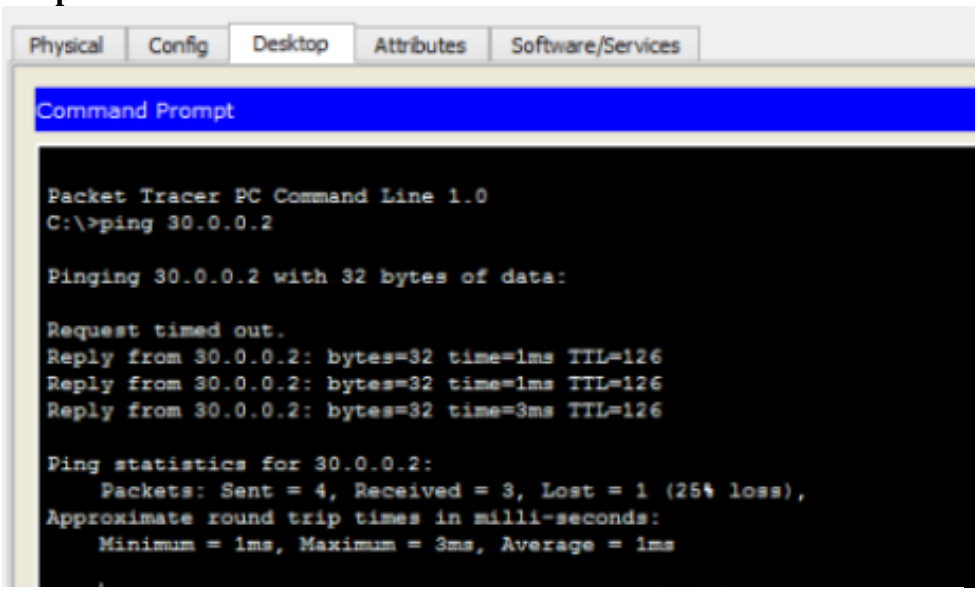
The configuration is pretty simple and requires only two major steps:

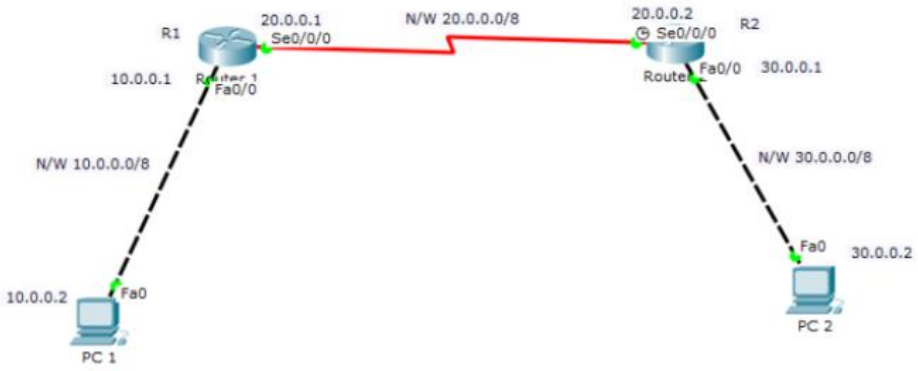
```
R1(config)#
R1(config)#router ospf 1
R1(config-router)#network 10.0.0.0 0.255.255.255 area 0
R1(config-router)#network 20.0.0.0 0.255.255.255 area 0
R2(config)#
R2(config)#router ospf 2
R2(config-router)#network 20.0.0.0 0.255.255.255 area 0
R2(config-router)#network 30.0.0.0 0.255.255.255 area 0
```

Verify OSPF configuration

```
R1#
R1#show ip ospf neighbor


Neighbor ID     Pri   State           Dead Time   Address
Interface
30.0.0.1          0   FULL/  -        00:00:30    20.0.0.2
Serial0/0/0

R1#
R1#show ip route ospf
O    30.0.0.0 [110/65] via 20.0.0.2, 00:20:50, Serial0/0/0
```

Output:





Observation	
Record	
Total	
Staff Signature	

Result:

Thus the OSPF configuration using packet tracer is established and verified.

EX.NO. 10B DATE: 12/11/2021	RIP CONFIGURATION USING PACKET TRACER

Aim:

To establish RIP (Routing Information Protocol) configuration between the end devices.

Algorithm:

1. Configure IP addresses on the PCs and the routers.
2. Configure the end devices.
3. Establish connection from source to destination through routers.
4. Enable the router configuration.
5. Find the best path from source to destination.

Commands:

ROUTER 1:

```
R1(config)#int fa 0/0
R1(config-if)#ip add 10.0.0.1 255.0.0.0
R1(config-if)#no shut
R1(config-if)#
R1(config-if)#int serial 0/0/0
R1(config-if)#ip add 20.0.0.1 255.0.0.0
R1(config-if)#no shut
```

ROUTER 2:

```
R2(config-if)#int fa0/0
R2(config-if)#ip add 30.0.0.1 255.0.0.0
R2(config-if)#no shut
R2(config-if)#
R2(config-if)#int serial0/0/0
R2(config-if)#ip address 20.0.0.2 255.0.0.0
R2(config-if)#no shut
```

IP configuration on PCs:

PC1 IP add 10.0.0.2 Subnet mask 255.0.0.0 Default gateway 10.0.0.1
PC2 IP add 30.0.0.2 Subnet mask 255.0.0.0 Default gateway 30.0.0.1

Configure RIPV2 on the routers.

The configuration is pretty simple and requires only two major steps:

ROUTER 1

```
R1(config)#
R1(config-router)#router rip
R1(config-router)#version 2
R1(config-router)#network 10.0.0.0
R1(config-router)#network 20.0.0.0
```

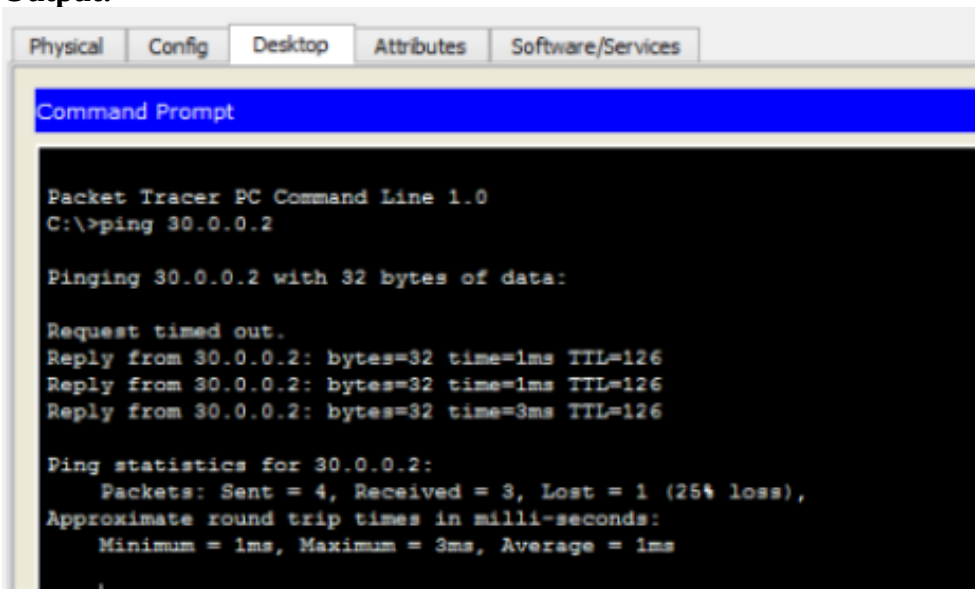
ROUTER 2
R2(config)#
R1(config-router)#router rip
R1(config-router)#version 2
R2(config-router)#network 20.0.0.0
R2(config-router)#network 30.0.0.0

Verify RIP configuration

To verify that RIP is indeed advertising routes, we can use the show ip-route commands on R1.

```
R1#
R1#show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile,
B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter
area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external
type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E -
EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia -
IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route

Gateway of last resort is not set

C    10.0.0.0/8 is directly connected, FastEthernet0/0
C    20.0.0.0/8 is directly connected, Serial0/0/0
R    30.0.0.0/8 [120/1] via 20.0.0.2, 00:00:17, Serial0/0/0
```

Output:



Observation	
Record	
Total	
Staff Signature	

Result:

Thus the router configuration using packet tracer is performed and verified.

EX.NO. 2	APPLICATIONS USING TCP SOCKETS

EX.NO. 2A DATE: 09/09/2021	ECHO CLIENT & ECHO SERVER USING TCP SOCKETS

Aim:

To implement the java program for echo client and echo server using TCP sockets.

Algorithm:

Client side:

1. Start the program.
2. Create a socket which binds the IP address of server and the port address to acquire service.
3. after establishing connection send a data to server.
4. Receive and print the same data from server.
5. Close the socket.
6. End the program.

Server side:

1. Start the program.
2. Create a server socket to activate the port address.
3. Create a socket for the server socket which accepts the connection.
4. after establishing connection receive the data from client.
5. Print and send the same data to client.
6. Close the socket.
7. End the program.

Program:

Server side:

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

public class EchoServer {
   private Socket socket=null;
   private ServerSocket server=null;
   private DataInputStream in=null;
   public EchoServer(int port){
      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: ");
            in=new DataInputStream(new
BufferedInputStream(socket.getInputStream()));
            String line="";
            while(!line.equals("Over"))
            {
               try{
                  line=in.readUTF();
                  System.out.println(line);
               }
               catch(IOException e){
                  System.out.println(e);
               }
            }
            System.out.println("Closing connection.");
            socket.close();
            in.close();
         }
         catch(IOException i)
         {
            System.out.println(i);
         }
      }
      public static void main(String args[])
      {
         EchoServer server=new EchoServer(5000);
      }
}
```

Client side:

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

public class EchoClient {
   private Socket socket=null;
   private DataInputStream input=null;
   private DataOutputStream out=null;
   public EchoClient(String address,int port)
   {
      try
      {
```

```
            socket=new Socket(address,port);
            System.out.println("Connected with server");
            input=new DataInputStream(System.in);
            out=new DataOutputStream(socket.getOutputStream());
        }
        catch(UnknownHostException u)
        {
            System.out.println(u);
        }
        catch(IOException i)
        {
            System.out.println(i);
        }
        String line="";
        while(!line.equals("Over"))
        {
            try
            {
                line=input.readLine();
                out.writeUTF(line);
            }
            catch(IOException i)
            {
                System.out.println(i);
            }
        }
        try
        {
            input.close();
            out.close();
            socket.close();
        }
        catch(IOException i)
        {
            System.out.println(i);
        }
    }
    public static void main(String args[])
    {
        EchoClient client=new EchoClient("127.0.0.1",5000);
    }
}
```

Output:

Server:

```
Server started.
Waiting for a client.....
Client accepted:
Hi server
I hope the connection is successfully done
Bye server
Over
Closing connection.
BUILD SUCCESSFUL (total time: 32 seconds)
```

Client:

```
run:
Connected with server
Hello Server
I hope the connection is successfully done!
Bye server
Over
BUILD SUCCESSFUL (total time: 31 seconds)
```

Result:

Thus the java program for echo client and echo server using TCP sockets are written and executed successfully.

EX.NO. 2B DATE: 09/09/2021	CHAT APPLICATION USING TCP SOCKETS

Aim:

To write a client-server application for chat using TCP sockets.

Algorithm:

Client side:

8. Start the program.
9. Include necessary package in java.
10. To create a socket in client to server.
11. The client establishes a connection to the server.
12. The client accepts the connection and to send the data from client to server.
13. The client communicates the server to send the end of the message.
14. Stop the program.

Server side:

8. Start the program.
9. Include necessary package in java.
10. To create a socket in server to client.
11. The server establishes a connection to the client.
12. The server accept the connection and to send the data from server to client and vice versa.
13. The server communicates the client to send the end of the message.
14. Stop the program.

Program:

Server side:

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

public class ChatServer {
   public static void main(String[] args)throws Exception
   {
      ServerSocket ss=new ServerSocket(2000);
      Socket sk=ss.accept();
      BufferedReader cin=new BufferedReader(new
InputStreamReader(sk.getInputStream()));
      PrintStream cout=new PrintStream(sk.getOutputStream());
      BufferedReader stdin=new BufferedReader(new
InputStreamReader(System.in));
```

```
        String s;
        while(true)
        {
            s=cin.readLine();
            if(s.equalsIgnoreCase("End"))
            {
                cout.println("Bye");
                break;
            }
            System.out.println("Client: "+s);
            System.out.print("Server: ");
            s=stdin.readLine();
            cout.println(s);
        }
        ss.close();
        sk.close();
        cin.close();
        cout.close();
    }
}
```

Client side:

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

public class ChatClient {
    public static void main(String[] args)throws Exception
    {
        Socket sk=new Socket("127.0.0.1",2000);
        BufferedReader sin=new BufferedReader(new
InputStreamReader(sk.getInputStream()));
        PrintStream sout=new PrintStream(sk.getOutputStream());
        BufferedReader stdin=new BufferedReader(new
InputStreamReader(System.in));
        String s;
        while(true)
        {
            System.out.print("Client: ");
            s=stdin.readLine();
            sout.println(s);
            s=sin.readLine();
            System.out.println("Server: "+s);
```

```
            if(s.equalsIgnoreCase("Bye"))
                break;
        }
        sk.close();
        sin.close();
        sout.close();
        stdin.close();
    }
}
```

Output:

Server:

```
Client: Hi server
Server: Hi client. How can I help you?
Client: I am just checking if the connection is successful.
Server: Yes, of course. It is successfully connected.
Client: That's great. Thank you server. Bye
Server: It's my pleasure client. Bye and take care.
End
```

Client:

```
Client: Hi server
Server: Hi client. How can I help you?
Client: I am just checking if the connection is successful.
Server: Yes, of course. It is successfully connected.
Client: That's great. Thank you server. Bye
Server: It's my pleasure client. Bye and take care.
Client: Bye
```

Result:

Thus the java program for chat application between server and client using TCP sockets are written and executed successfully.

EX.NO. 2C DATE: 09/09/2021	FILE TRANSFER BETWEEN CLIENT & SERVER USING TCP SOCKETS

Aim:

To perform file transfer between client and server using TCP sockets.

Algorithm:

Client side:

1. Start.
2. Establish a connection between the Client and Server.
3. Sockets s = new Socket(InetAddress. getLocalHost(), 1100);
4. Implement a client that can send two requests.
    i) To get a file from the server.
    ii) To put or send a file to the server.
5. After getting approval from the server, the clients either get file from the server or send file to the server.

Server side:

1. Start.
2. Implement a server socket that listens to a particular port number.
3. Server reads the filename and sends the data stored in the file for the‘get’ request.
4. It reads the data from the input stream and writes it to a file in the server for the ‘put’ instruction.
5. Exit upon client’s request.
6. Stop.

Program:

Server side:

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

public class fileserver {
   public static void main(String[] args)throws Exception{
      ServerSocket ssock=new ServerSocket(5000);
      Socket socket=ssock.accept();
      InetAddress ia=InetAddress.getByName("localhost");
      File f=new File("D:\\Sonal College\\5th sem files\\Computer
Networks\\Lab\\fileTransfer.txt");
      FileInputStream fin=new FileInputStream(f);
      BufferedInputStream bis=new BufferedInputStream(fin);
      OutputStream os=socket.getOutputStream();
```

```
        byte[] contents;
        long fileLength=f.length();
        long current=0;
        long start=System.nanoTime();
        while(current!=fileLength)
        {
           int size=10000;
           if(fileLength-current>=size)
              current+=size;
           else
           {
              size=(int)(fileLength-current);
              current=fileLength;
           }
           contents=new byte[size];
           bis.read(contents,0,size);
           os.write(contents);
           System.out.println("Sending file....."+(current*100)/fileLength+"%
complete");
        }
        os.flush();
        socket.close();
        ssock.close();
        System.out.println("File sent successfully!");
    }
}
```

Client side:

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

public class fileclient {
    public static void main(String[] args)throws Exception{
        Socket socket=new Socket(InetAddress.getByName("localhost"),5000);
        byte[] contents=new byte[10000];
        FileOutputStream fout=new FileOutputStream("D:\\lrsub\\Sonal
College\\5th sem files\\Computer Networks\\Lab\\fileTransfer1.txt");
        BufferedOutputStream bos=new BufferedOutputStream(fout);
        InputStream is=socket.getInputStream();
        int bytesRead=0;
        while((bytesRead=is.read(contents))!=-1)
        {
```

```
            bos.write(contents,0,bytesRead);
        }
        bos.flush();
        socket.close();
        System.out.println("File saved successfully!");
    }
}
```

Output:

Server:

```
Sending file.....100% complete
File sent successfully!
BUILD SUCCESSFUL (total time: 11 seconds)
```

Client:

```
File saved successfully!
BUILD SUCCESSFUL (total time: 1 second)
```

Result:

Thus the java program for file transfer between client and server using TCP sockets are written and executed successfully.

EX.NO. 3 DATE: 09/09/2021	HTTP WEB CLIENT PROGRAM TO DOWNLOAD A WEB PAGE USING TCP SOCKETS

Aim:

To download a webpage using TCP sockets

Algorithm:

Client side:

1. Start the program.
2. Create a socket which binds the Ip address of server and the port address to acquire service.
3. After establishing connection send the url to server.
4. Open a file and store the received data into the file.
5. Close the socket.
6. End the program.

Server side:

1. Start the program.
2. Create a server socket to activate the port address.
3. Create a socket for the server socket which accepts the connection.
4. After establishing connection receive url from client.
5. Download the content of the url received and send the data to client.
6. Close the socket.
7. End the program.

Program:

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;

public class urldownload {
   private static final int BUFFER_SIZE = 4096;
   public static void downloadFile(String fileURL, String saveDir)
   throws IOException {
      URL url = new URL(fileURL);
      HttpURLConnection httpConn = (HttpURLConnection)
url.openConnection();
      int responseCode = httpConn.getResponseCode();
      if (responseCode == HttpURLConnection.HTTP_OK) {
```

```
            String fileName = "";
            String disposition = httpConn.getHeaderField("Content-Disposition");
            String contentType = httpConn.getContentType();
            int contentLength = httpConn.getContentLength();
            if (disposition != null) {
               int index = disposition.indexOf("filename=");
               if (index > 0) {
                  fileName = disposition.substring(index + 10,
                        disposition.length() - 1);
               }
            }
            else {
               fileName = fileURL.substring(fileURL.lastIndexOf("/") + 1,
                     fileURL.length());
            }
            System.out.println("Content-Type = " + contentType);
            System.out.println("Content-Disposition = " + disposition);
            System.out.println("Content-Length = " + contentLength);
            System.out.println("fileName = " + fileName);
            InputStream inputStream = httpConn.getInputStream();
            String saveFilePath = saveDir + File.separator + fileName;
            FileOutputStream outputStream = new FileOutputStream(saveFilePath);
            int bytesRead = -1;
            byte[] buffer = new byte[BUFFER_SIZE];
            while ((bytesRead = inputStream.read(buffer)) != -1) {
               outputStream.write(buffer, 0, bytesRead);
            }
            outputStream.close();
            inputStream.close();
            System.out.println("File downloaded");
         }
           else {
            System.out.println("No file to download. Server replied HTTP code: " +
responseCode);
         }
         httpConn.disconnect();
      }
      public static void main(String[] args) {
         String fileURL = "
https://www.abc.net.au/reslib/200709/r186930_1738931.jpg ";
         String saveDir = " D:\Sonal College\5th sem files\Computer Networks ";
         try {
            http.downloadFile(fileURL, saveDir);
```

```
        }
          catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

Output:

```
run:
Content-Type=image/jpeg
Content-Disposition=null
Content-Length=86851
Filename=r186930_1738931.jpg
File downloaded
BUILD SUCCESSFUL (total time: 0 seconds)
```

Downloaded file:

Result:

Thus the java program for HTTP web client to download a webpage using TCP sockets were written and successfully executed.

EX.NO. 4 DATE: 20/09/2021	SIMULATION OF ARP AND RARP PROTOCOL

4A. Address Resolution Protocol (ARP):

Aim:

To simulate Address Resolution Protocol (ARP) using UDP.

Algorithm:

Client side:

1. Establish a connection between the client and server.
2. Get the IP address to be connected into MAC address.
3. Send this IP address to the server .
4. Server returns the Mac address to the client.
6. End the program.

Server side:

1. Start the program.
2. Accept the socket which is created by the client.
3. Server maintains the table in which IP and the corresponding MAC address are stored.
4. Read the IP address which is sent by the client.
5. Map the IP address with it's MAC address and return the Mac address to the client.
5. End the program.

Program:

Client side:

ARP CLIENT

```
import java.io.*;
import java.net.*;
import java.util.*;
public class ARPClient{
   public static void main(String args[]){
      try{
         DatagramSocket client=new DatagramSocket();
         InetAddress addr=InetAddress.getByName("127.0.0.1");
         byte[] sendbyte=new byte[1024];
         byte[] receivebyte=new byte[1024];
         BufferedReader in=new BufferedReader(new
InputStreamReader(System.in));
         System.out.println("Enter the IP address: ");
         String str=in.readLine();
         sendbyte=str.getBytes();
```

```
            DatagramPacket sender=new
DatagramPacket(sendbyte,sendbyte.length,addr,1309);
            client.send(sender);
            DatagramPacket receiver=new
DatagramPacket(receivebyte,receivebyte.length);
            client.receive(receiver);
            String s=new String(receiver.getData());
            System.out.println("The physical address is "+s.trim());
            client.close();
        }
        catch(Exception e) {
            System.out.println(e);
        }
    }
}
```

Server side:

ARP SERVER

```
import java.io.*;
import java.net.*;
import java.util.*;
class ARPServer{
   public static void main(String args[]){
      try
         DatagramSocket server=new DatagramSocket(1309);
         while(true) {
            byte[] sendbyte=new byte[1024];
            byte[] receivebyte=new byte[1024];
            DatagramPacket receiver=new
DatagramPacket(receivebyte,receivebyte.length);
            server.receive(receiver);
            String str=new String(receiver.getData());
            String s=str.trim();
                        //System.out.println(s);
            InetAddress addr=receiver.getAddress();
            int port=receiver.getPort();
            String ip[]={"165.165.80.80","165.165.79.1"};
            String mac[]={"6A:08:AA:C2","8A:BC:E3:FA"};
            for(int i=0;i<ip.length;i++) {
               if(s.equals(ip[i]))  {
                  sendbyte=mac[i].getBytes();
                  DatagramPacket sender=new
DatagramPacket(sendbyte,sendbyte.length,addr,port);
```

```
                server.send(sender);
                break;
              }
            }
           break;
        }
      }
      catch(Exception e) {
         System.out.println(e);
      }
   }
}
```

Output:

```
Enter the IP address:
165.165.79.1
The physical address is 8A:BC:E3:FA
BUILD SUCCESSFUL (total time: 13 seconds)
```

4B. Reverse Address Resolution Protocol (RARP):

Aim:

To write a java program for simulating Reverse Address Resolution Protocol (RARP) using UDP.

Algorithm:

Client side:

1. Start the program.
2. Using datagram sockets UDP function is established.
3. Get the MAC address to server.
4. Send this MAC address to server.
5. Server returns the IP address to the client.
6. Stop the program.

Server side:

1. Start the program.
2. Server maintains the table in which IP and the corresponding MAC address are stored.
3. Read the MAC address which is sent by the client.
4. Map the IP address with it's MAC address and return the IP address to the client.
5. End the program.

Program:

Client side:

RARP CLIENT

```
import java.io.*;
import java.net.*;
import java.util.*;
public class RARPClient {
   public static void main(String[] args) {
      try {
         DatagramSocket client=new DatagramSocket();
         InetAddress addr=InetAddress.getByName("127.0.0.1");
         byte[] sendbyte=new byte[1024];
         byte[] receivebyte=new byte[1024];
         BufferedReader in=new BufferedReader(new
InputStreamReader(System.in));
         System.out.println("Enter the physical address: ");
         String str=in.readLine();
         sendbyte=str.getBytes();
         DatagramPacket sender=new DatagramPacket(sendbyte,
sendbyte.length, addr, 1309);
         client.send(sender);
         DatagramPacket receiver=new DatagramPacket(receivebyte,
receivebyte.length);
         client.receive(receiver);
         String s=new String(receiver.getData());
         System.out.println("The IP address is "+s.trim());
         client.close();
      }
      catch(Exception e) {
         System.out.println(e);
      }
   }
}
```

Server side:

RARP SERVER

```
import java.io.*;
import java.net.*;
import java.util.*;
public class RARPServer {
   public static void main(String[] args) {
      try {
         DatagramSocket server=new DatagramSocket(1309);
```

```
        while(true) {
            byte[] sendbyte=new byte[1024];
            byte[] receivebyte=new byte[1024];
            DatagramPacket receiver=new DatagramPacket(receivebyte,
receivebyte.length);
            server.receive(receiver);
            String str=new String(receiver.getData());
            String s=str.trim();
            InetAddress addr=receiver.getAddress();
            int port=receiver.getPort();
            String ip[]={"165.165.80.80", "165.165.79.1"};
            String mac[]={"68:08:AA:C2", "8A:BC:E3:FA"};
            for(int i=0; i<ip.length; i++) {
               if(s.equals(mac[i])) {
                  sendbyte=ip[i].getBytes();
                  DatagramPacket sender=new DatagramPacket(sendbyte,
sendbyte.length, addr, port);
                  server.send(sender);
                  break;
               }
            }
            break;
         }
      }
      catch(Exception e) {
         System.out.println(e);
      }
   }
}
```

Output:

```
Enter the physical address:
68:08:AA:C2
The IP address is 165.165.80.80
BUILD SUCCESSFUL (total time: 9 seconds)
```

Result:

Thus the java program for simulation of ARP and RARP protocols using UDP were written and successfully executed.

EX.NO. 5 DATE: 20/10/2021	SIMULATION OF ERROR DETECTION USING CRC

Aim:

To write a java program for the simulation of error detection using CRC.

Algorithm:

1. Open the editor and type the program for error detection
2. Get the input in the form of bits.
3. Append the redundancy bits.
4. Divide the appended data using a divisor polynomial.
5. The resulting data should be transmitted to the receiver.
6. At the receiver the received data is entered.
7. The same process is repeated at the receiver.
8. If the remainder is zero there is no error otherwise there is some error in the received bits.
9. Run the program.

Program:

```
import java.io.*;
public static void main(String args[]) throws IOException
{
 BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
 System.out.println("Enter Generator:");
 String gen = br.readLine();
 System.out.println("Enter Data:");
 String data = br.readLine();
 String code = data;
 while(code.length() < (data.length() + gen.length() - 1))
  code = code + "0";
  code = data + div(code,gen);
 System.out.println("The transmitted Code Word is: " + code);
 System.out.println("Please enter the received Code Word: ");
 String rec = br.readLine();
 if(Integer.parseInt(div(rec,gen)) == 0)
  System.out.println("The received code word contains no errors.");
 else
  System.out.println("The received code word contains errors.");
}

static String div(String num1,String num2)
{
 int pointer = num2.length();
 String result = num1.substring(0, pointer);
 String remainder = "";
 for(int i = 0; i < num2.length(); i++)
 {
  if(result.charAt(i) == num2.charAt(i))
   remainder += "0";
  else
   remainder += "1";
```

```
 }
 while(pointer < num1.length())
 {
  if(remainder.charAt(0) == '0')
  {
   remainder = remainder.substring(1, remainder.length());
   remainder = remainder + String.valueOf(num1.charAt(pointer));
   pointer++;
  }
  result = remainder;
  remainder = "";
  for(int i = 0; i < num2.length(); i++)
  {
   if(result.charAt(i) == num2.charAt(i))
    remainder += "0";
   else
    remainder += "1";
  }
 }
 return remainder.substring(1,remainder.length());
}
```

Output:

```
run:
Enter Generator:
1101
Enter Data:
100100
The transmitted Code Word is: 100100001
Please enter the received Code Word:
100100001
The received code word contains no errors.
BUILD SUCCESSFUL (total time: 33 seconds)
```

Result:

Thus the java program for simulation of error detection using CRC were written and successfully executed.

EX.NO. 6 DATE: 20/10/2021	IMPLEMENTATION OF DISTANCE VECTOR ROUTING AND LINK STATE ROUTING ALGORITHM

Aim:

To simulate and observe traffic route of a network using distance vector and link state routing algorithm.

Algorithm:

17. Create a simulator object.
18. Set routing protocol to distance vector routing.
19. Track packets on all links on to NAM trace and text trace file.
20. Define finish procedure to close files, flash tracing and run NAM.
21. Create five nodes.
22. Specify the link characteristics between the nodes.
23. Describe their layout topology as an octagon.
24. Add UDP agent for node n_o.
25. Create CBR traffic on the top of the UDP and set the traffic traffic parameters.
26. Add NULL agent to node n_3.
27. Connect source and sink.
28. Schedule as follows:
    - Start traffic flow at 1.0
    - Down the link n_1-n_2 at 15.0
    - Up the link n_1-n_2 at 25.0
    - Call finish procedure at 35.0
29. Start the schedule.
30. Observe the traffic route when the link is up and down.
31. View the simulated events and trace file, analyze it.
32. Stop.

6A. Distance Vector Routing:

Program:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Distance_Vectore_Routing {
   static int graph[][];
   static int via[][];
   static int rt[][];
   static int v;
   static int e;
public static void main(String args[]) throws IOException{
  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
  System.out.println("Enter the number of Vertices: ");
  v = Integer.parseInt(br.readLine());
  System.out.println("Enter the number of Edges: ");
  e = Integer.parseInt(br.readLine());
  graph = new int[v][v];
  via = new int[v][v];
  rt = new int[v][v];
```

```
  for(int i = 0; i < v; i++)
  for(int j = 0; j < v; j++)
  {
    if(i == j)
      graph[i][j] = 0;
    else
      graph[i][j] = 9999;
  }
  for(int i = 0; i < e; i++)
 {
    System.out.println("Enter data for Edge " + (i + 1) + ":");
    System.out.print("Source: ");
    int s = Integer.parseInt(br.readLine());
    s--;
    System.out.print("Destination: ");
    int d = Integer.parseInt(br.readLine());
    d--;
    System.out.print("Cost: ");
    int c = Integer.parseInt(br.readLine());
    graph[s][d] = c;
    graph[d][s] = c;
  }
  dvr_calc_disp("The initial Routing Tables are: ");
  System.out.print("Please enter the Source Node for the edge whose cost has changed: ");
  int s = Integer.parseInt(br.readLine());
  s--;
  System.out.print("Please enter the Destination Node for the edge whose cost has changed:
");
  int d = Integer.parseInt(br.readLine());
  d--;
  System.out.print("Please enter the new cost: ");
  int c = Integer.parseInt(br.readLine());
  graph[s][d] = c;
  graph[d][s] = c;
 dvr_calc_disp("The new Routing Tables are: ");
 }
 static void dvr_calc_disp(String message)
{
  System.out.println();
  init_tables();
  update_tables();
  System.out.println(message);
  print_tables();
  System.out.println();
}

static void update_table(int source)
{
  for(int i = 0; i < v; i++)
  {
```

```
   if(graph[source][i] != 9999)
  {
     int dist = graph[source][i];
     for(int j = 0; j < v; j++)
     {
        int inter_dist = rt[i][j];
        if(via[i][j] == source)
          inter_dist = 9999;
        if(dist + inter_dist < rt[source][j])
        {
          rt[source][j] = dist + inter_dist;
          via[source][j] = i;
        }
     }
  }
 }
}

static void update_tables()
{
  int k = 0;
  for(int i = 0; i < 4*v; i++)
  {
     update_table(k);
     k++;
     if(k == v)
     k = 0;
  }
 }

static void init_tables()
{
   for(int i = 0; i < v; i++)
  {
     for(int j = 0; j < v; j++)
     {
        if(i == j)
        {
          rt[i][j] = 0;
          via[i][j] = i;
        }
       else
       {
         rt[i][j] = 9999;
         via[i][j] = 100;
        }
     }
   }
 }
```

```
static void print_tables()
{
   for(int i = 0; i < v; i++)
   {
      for(int j = 0; j < v; j++)
     {
         System.out.print("Dist: " + rt[i][j] + "    ");
      }
      System.out.println();
    }
  }
}
```

Output:

```
run:
Enter the number of Vertices:
4
Enter the number of Edges:
5
Please enter data for Edge 1:
Source: 1
Destination: 3
Cost: 3
Please enter data for Edge 2:
Source: 1
Destination: 2
Cost: 4
Please enter data for Edge 3:
Source: 1
Destination: 4
Cost: 3
Please enter data for Edge 4:
Source: 2
Destination: 4
Cost: 3
Please enter data for Edge 5:
Source: 2
Destination: 3
Cost: 5
The initial Routing Tables are:
Dist: 0    Dist: 4    Dist: 3    Dist: 3
Dist: 4    Dist: 0    Dist: 5    Dist: 3
Dist: 3    Dist: 5    Dist: 0    Dist: 6
Dist: 3    Dist: 3    Dist: 6    Dist: 0

Please enter the Source Node for the edge whose cost has changed: 2
Please enter the Destination Node for the edge whose cost has changed: 3
Please enter the new cost: 1

The new Routing Tables are:
Dist: 0    Dist: 4    Dist: 3    Dist: 3
Dist: 4    Dist: 0    Dist: 1    Dist: 3
Dist: 3    Dist: 1    Dist: 0    Dist: 4
Dist: 3    Dist: 3    Dist: 4    Dist: 0

BUILD SUCCESSFUL (total time: 1 minute 10 seconds)
```

6B. Link State Routing:

Program:

```
import java.util.Scanner;
public class Link_State_Routing {
    public  int distance[] = new int[10];
    public  int cost[][] = new int[10][10];
    public void calc(int n,int s)
   {
      int flag[] = new int[n+1];
      int i,minpos=1,k,c,minimum;
      for(i=1;i<=n;i++)
     {
         flag[i]=0;
         this.distance[i]=this.cost[s][i];
     }
      c=2;
      while(c<=n)
     {
        minimum=99;
        for(k=1;k<=n;k++)
       {
           if(this.distance[k]<minimum && flag[k]!=1)
          {
            minimum=this.distance[i];
            minpos=k;
          }
       }
        flag[minpos]=1;
        c++;
        for(k=1;k<=n;k++)
       {
           if(this.distance[minpos]+this.cost[minpos][k] <  this.distance[k] && flag[k]!=1 )
             this.distance[k]=this.distance[minpos]+this.cost[minpos][k];
       }
     }
  }
  public static void main(String args[])
 {
    int nodes,source,i,j;
    Scanner in = new Scanner(System.in);
    System.out.println("Enter the Number of Nodes \n");
    nodes = in.nextInt();
    Link_State_Routing d = new Ex6A_Link_State_Routing();
    System.out.println("Enter the Cost Matrix Weights: \n");
    for(i=1;i<=nodes;i++)
    for(j=1;j<=nodes;j++)
    {
      d.cost[i][j]=in.nextInt();
      if(d.cost[i][j]==0)
      d.cost[i][j]=999;
```

```
    }
    System.out.println("Enter the Source Vertex :\n");
    source=in.nextInt();
    d.calc(nodes,source);
    System.out.println("The Shortest Path from Source \t"+source+"\t to all other vertices are :
\n");
    for(i=1;i<=nodes;i++)
      if(i!=source)
        System.out.println("source :"+source+"\t destination :"+i+"\t MinCost is
:"+d.distance[i]+"\t");
   }
}
```

Output:

```
run:
Enter the Number of Nodes

5
Enter the Cost Matrix Weights:

0 1 0 2 0
1 0 1 0 4
0 1 0 0 3
2 0 0 0 3
0 4 3 3 0
Enter the Source Vertex :

1
The Shortest Path from Source   1       to all other vertices are :

source :1       destination :2  MinCost is :1
source :1       destination :3  MinCost is :2
source :1       destination :4  MinCost is :2
source :1       destination :5  MinCost is :5
BUILD SUCCESSFUL (total time: 53 seconds)
```

Observation	
Record	
Total	
Staff Signature	

Result:

Thus the java program to simulate and observe traffic route of a network using distance vector and link state routing algorithm were written and successfully executed.

EX.NO. 7 DATE: 28/10/2021	SIMULATION OF DNS USING UDP SOCKETS

Aim:

To write a java program for the simulation of DNS using UDP sockets.

Algorithm:

1. Start the program.
2. Get the frame size from the user.
3. To create the frame based on the user request.
4. To send frames to server from the client side.
5. If your frames reach the server, it will send ACK signal to client otherwise it will send NACK signal to the client.
6. Stop the program.

Program:

UDP DNS Server:

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

public class UDP_DNS_Server {
   private static int indexOf(String[] array, String str){
      str = str.trim();
      for (int i=0; i < array.length; i++)
      {
         if (array[i].equals(str)) return i;
      }
      return -1;
   }
   public static void main(String arg[])throws IOException{
      String[] hosts = {"zoho.com", "gmail.com","google.com", "facebook.com"};
      String[] ip = {"172.28.251.59", "172.217.11.5","172.217.11.14","31.13.71.36"};
      System.out.println("Press Ctrl + C to Quit");
      while (true){
         DatagramSocket serversocket=new DatagramSocket(1362);
         byte[] senddata = new byte[1021];
         byte[] receivedata = new byte[1021];
         DatagramPacket recvpack = new DatagramPacket(receivedata,
         receivedata.length);
         serversocket.receive(recvpack);
         String sen = new String(recvpack.getData());
         InetAddress ipaddress = recvpack.getAddress();
         int port = recvpack.getPort();
         String capsent;
         System.out.println("Request for host " + sen);
         if(indexOf (hosts, sen) != -1)
            capsent = ip[indexOf (hosts, sen)];
         else
```

```
            capsent = "Host Not Found"; senddata = capsent.getBytes();
          DatagramPacket pack = new DatagramPacket (senddata,
          senddata.length,ipaddress,port);
          serversocket.send(pack);
          serversocket.close();
       }
    }
}
```

UDP DNS Client:

```
public class UDP_DNS_Client {
   public static void main(String args[])throws IOException{
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      DatagramSocket clientsocket = new DatagramSocket();
      InetAddress ipaddress;
      if (args.length == 0)
         ipaddress = InetAddress.getLocalHost();
      else
         ipaddress = InetAddress.getByName(args[0]);
      byte[] senddata = new byte[1024];
      byte[] receivedata = new byte[1024];
      int portaddr = 1362;
      System.out.print("Enter the hostname : ");
      String sentence = br.readLine();
      senddata = sentence.getBytes();
      DatagramPacket pack = new
DatagramPacket(senddata,senddata.length,ipaddress,portaddr);
      clientsocket.send(pack);
      DatagramPacket recvpack =new DatagramPacket(receivedata,receivedata.length);
      clientsocket.receive(recvpack);
      String modified = new String(recvpack.getData());
      System.out.println("IP Address: " + modified);
      clientsocket.close();
   }
}
```

Output:

Server side:

```
run:
Press Ctrl + C to Quit
Request for host google.com
```

Client side:

```
run:
Enter the hostname : google.com
IP Address: 172.217.11.14
BUILD SUCCESSFUL (total time: 13 seconds)
```

Observation	
Record	
Total	
Staff Signature	

Result:

Thus the java program for the simulation of DNS using UDP sockets were written and successfully executed.

EX.NO. 8 DATE: 12/11/2021	SWITCH CONFIGURATION USING PACKET TRACER

Aim:

To establish a basic switch configuration between the end devices.

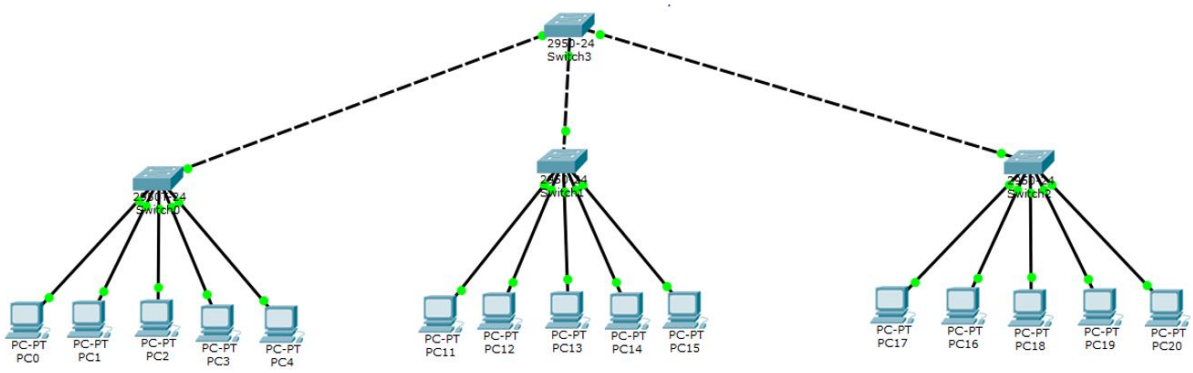
Algorithm:

1. Configure the end devices.
2. Establish connection from source to destination through switches.
3. Enable the switch.
4. Configure console password and vty password.

Commands:

```
Switch>enable
Switch#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Switch(config)#hostname cse
cse(config)#
cse (config)#line console 0
cse(config-line)#password lab1
cse (config-line)#login
cse (config-line)#line vty 0 15
cse (config-line)#password lab1
cse (config-line)#login
cse (config-line)#exit
```

Output:



Observation	
Record	
Total	
Staff Signature	

Result:

Thus the switch configuration using packet tracer is performed and verified.

EX.NO. 9 DATE: 12/11/2021	ROUTER CONFIGURATION USING PACKET TRACER

Aim:

To establish a basic router configuration between the end devices.

Algorithm:

1. Configure the end devices.
2. Establish connection from source to destination through switches.
3. Click the router and enter the code configuration in cli tab.

Commands:

ROUTER 1:

```
Router>enable
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface serial 2/0
Router(config-if)#ip address 172.16.1.1 255.255.0.0
Router(config-if)#no shutdown

Router(config-if)#
%LINK-5-CHANGED: Interface Serial 2/0, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface serial0/0, changed state to up
Router(config-if)#interface serial 2/0
Router(config-if)#ip address 172.16.1.3 255.255.0.0
Router(config-if)#no shutdown

Router(config-if)#
%LINK-5-CHANGED: Interface serial 2/0, changed state to up

Router#
%SYS-5-CONFIG_I: Configured from console by console
```

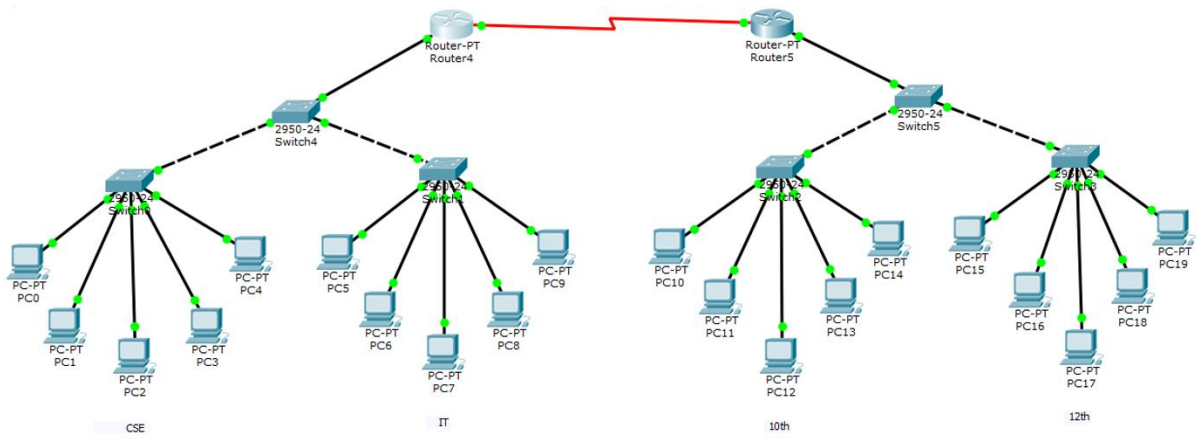
ROUTER 2:

```
Router>enable
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface serial 2/0
Router(config-if)#ip address 172.16.3.4 255.255.0.0
Router(config-if)#no shutdown

Router(config-if)#
%LINK-5-CHANGED: Interface serial 2/0, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface serial 2/0, changed state to up
Router(config)#interface serial 2/0
Router(config-if)#ip address 172.16.3.4  255.255.255.0
Router(config-if)#no shutdown
```

Router(config-if)#
%LINK-5-CHANGED: Interface serial 2/0, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface serial 2/0, changed state to up

Output:



Observation	
Record	
Total	
Staff Signature	

Result:

Thus the router configuration using packet tracer is performed and verified.

EX.NO. 10A DATE: 12/11/2021	OSPF CONFIGURATION USING PACKET TRACER

Aim:

To establish the OSPF (OPEN SHORTEST PATH FIRST) configuration between the end devices.

Algorithm:

1. Configure the end devices.
2. Configure connection from source to destination through routers
3. Enable the router configuration
4. Find the best path from source to destination.

Commands:

ROUTER 1:

```
R1(config)#int fa 0/0
R1(config-if)#ip add 10.0.0.1 255.0.0.0
R1(config-if)#no shut
R1(config-if)#
R1(config-if)#int serial 0/0/0
R1(config-if)#ip add 20.0.0.1 255.0.0.0
R1(config-if)#no shut
```

ROUTER 2:

```
R2(config-if)#int fa0/0
R2(config-if)#ip add 30.0.0.1 255.0.0.0
R2(config-if)#no shut
R2(config-if)#
R2(config-if)#int serial0/0/0
R2(config-if)#ip address 20.0.0.2 255.0.0.0
R2(config-if)#no shut
```

PC1 IP add 10.0.0.2 Subnet mask 255.0.0.0 Default gateway 10.0.0.1

PC2 IP add 30.0.0.2 Subnet mask 255.0.0.0 Default gateway 30.0.0.1

Configure OSPF on the routers.

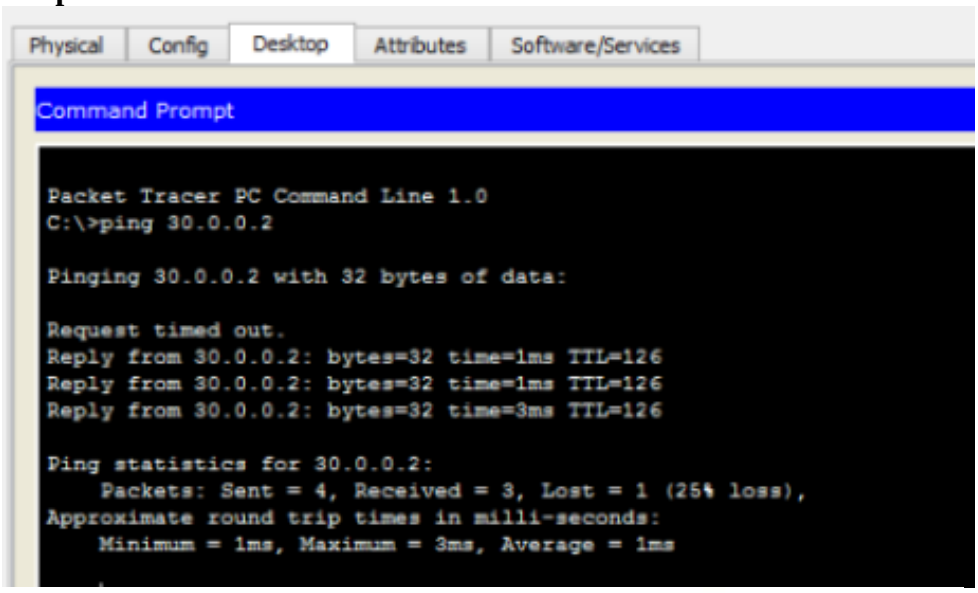
The configuration is pretty simple and requires only two major steps:

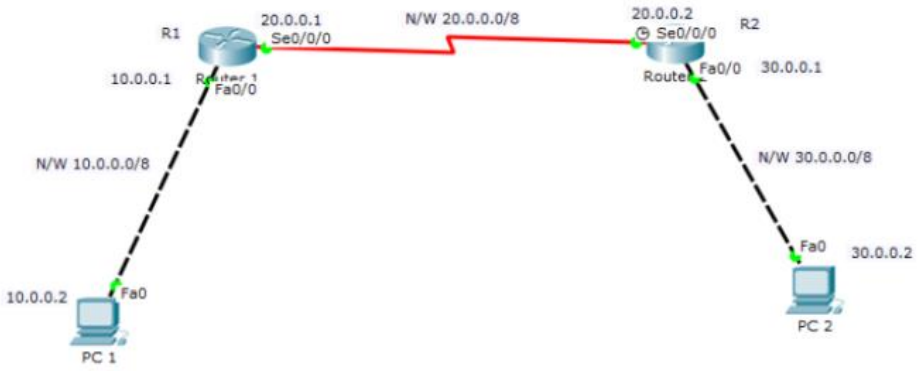
```
R1(config)#
R1(config)#router ospf 1
R1(config-router)#network 10.0.0.0 0.255.255.255 area 0
R1(config-router)#network 20.0.0.0 0.255.255.255 area 0
R2(config)#
R2(config)#router ospf 2
R2(config-router)#network 20.0.0.0 0.255.255.255 area 0
R2(config-router)#network 30.0.0.0 0.255.255.255 area 0
```

Verify OSPF configuration

```
R1#
R1#show ip ospf neighbor


Neighbor ID     Pri   State           Dead Time   Address
Interface
30.0.0.1          0   FULL/  -        00:00:30    20.0.0.2
Serial0/0/0

R1#
R1#show ip route ospf
O    30.0.0.0 [110/65] via 20.0.0.2, 00:20:50, Serial0/0/0
```

Output:





Observation	
Record	
Total	
Staff Signature	

Result:

Thus the OSPF configuration using packet tracer is established and verified.

EX.NO. 10B DATE: 12/11/2021	RIP CONFIGURATION USING PACKET TRACER

Aim:

To establish RIP (Routing Information Protocol) configuration between the end devices.

Algorithm:

1. Configure IP addresses on the PCs and the routers.
2. Configure the end devices.
3. Establish connection from source to destination through routers.
4. Enable the router configuration.
5. Find the best path from source to destination.

Commands:

ROUTER 1:

```
R1(config)#int fa 0/0
R1(config-if)#ip add 10.0.0.1 255.0.0.0
R1(config-if)#no shut
R1(config-if)#
R1(config-if)#int serial 0/0/0
R1(config-if)#ip add 20.0.0.1 255.0.0.0
R1(config-if)#no shut
```

ROUTER 2:

```
R2(config-if)#int fa0/0
R2(config-if)#ip add 30.0.0.1 255.0.0.0
R2(config-if)#no shut
R2(config-if)#
R2(config-if)#int serial0/0/0
R2(config-if)#ip address 20.0.0.2 255.0.0.0
R2(config-if)#no shut
```

IP configuration on PCs:

PC1 IP add 10.0.0.2 Subnet mask 255.0.0.0 Default gateway 10.0.0.1
PC2 IP add 30.0.0.2 Subnet mask 255.0.0.0 Default gateway 30.0.0.1

Configure RIPV2 on the routers.

The configuration is pretty simple and requires only two major steps:

ROUTER 1

```
R1(config)#
R1(config-router)#router rip
R1(config-router)#version 2
R1(config-router)#network 10.0.0.0
R1(config-router)#network 20.0.0.0
```

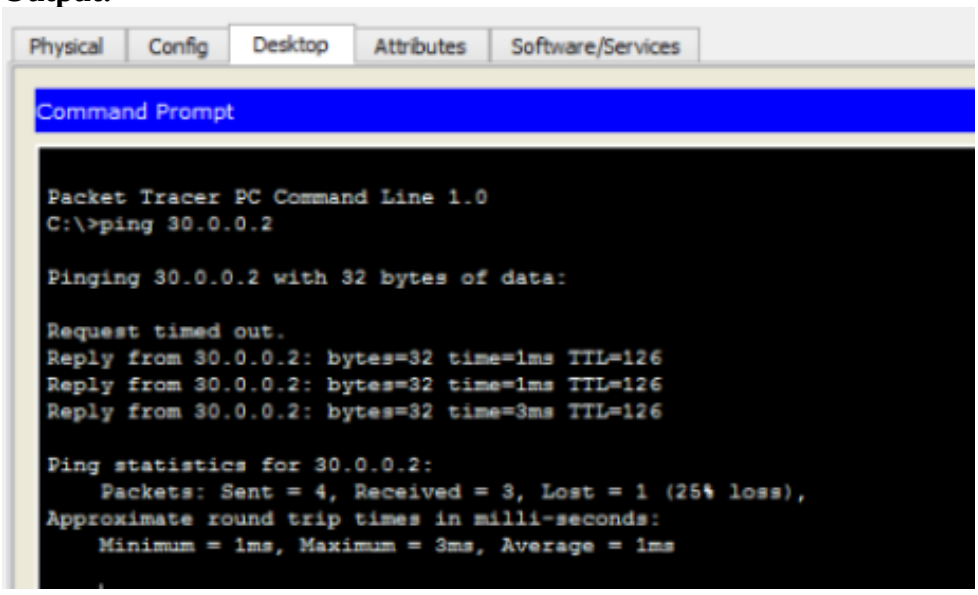
ROUTER 2
R2(config)#
R1(config-router)#router rip
R1(config-router)#version 2
R2(config-router)#network 20.0.0.0
R2(config-router)#network 30.0.0.0

Verify RIP configuration

To verify that RIP is indeed advertising routes, we can use the show ip-route commands on R1.

```
R1#
R1#show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile,
B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter
area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external
type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E -
EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia -
IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route

Gateway of last resort is not set

C    10.0.0.0/8 is directly connected, FastEthernet0/0
C    20.0.0.0/8 is directly connected, Serial0/0/0
R    30.0.0.0/8 [120/1] via 20.0.0.2, 00:00:17, Serial0/0/0
```

Output:



Observation	
Record	
Total	
Staff Signature	

Result:

Thus the router configuration using packet tracer is performed and verified.

EX.NO. 11 DATE: 03/12/2021	Study of Network Simulators - NS2

Introduction to NS-2 Simulator

NS is a discrete event simulator for networking research. It provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks.

Downloading/Installing ns

NS should configure and build on Unix systems. To install NS-2 in your own system, you can download the package from http://www.isi.edu/nsnam/ns/ns-build.html (To install NS in windows systems, please also refer this page). There are two ways to build ns: from the various packages or ‘ all-in-one’ package. For simplicity, it is recommended to start with the ‘ all-in-one’ package.

Build NS:

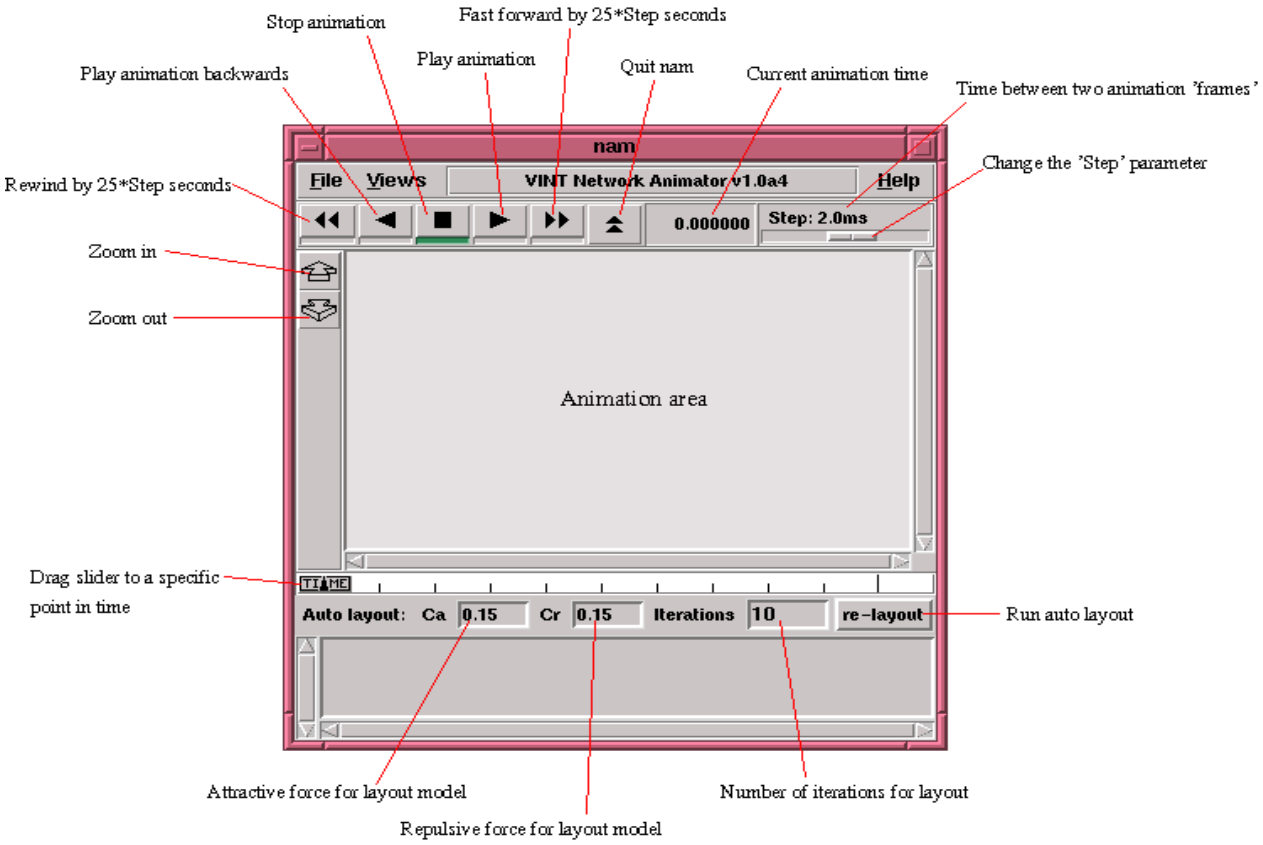
Entering NS2 directory and execute:
./gunzip ns-allinone-***.tar.gz /* *** is the version number */
/* unzip the file */
./tar xvf ns-allinone-***.tar /* untar the file */
./install
./validate
Please refer http://www.isi.edu/nsnam/ns/ns-problems.html for any installation problems.

Starting ns

You start ns with the command 'ns <tclscript>' (assuming that you are in the directory with the ns executable, or that your path points to that directory), where '<tclscript>' is the name of a Tcl script file which defines the simulation scenario (i.e. the topology and the events). You could also just start ns without any arguments and enter the Tcl commands in the Tcl shell, but that is definitely less comfortable. For information on how to write your own Tcl scripts for ns.Everything else depends on the Tcl script. The script might create some output on stdout, it might write a trace file or it might start nam to visualize the simulation. Or all of the above. These possibilities will all be discussed in later sections.

10.3 Starting nam

You can either start nam with the command 'nam <nam-file>' where '<nam-file>' is the name of a nam trace file that was generated by ns, or you can execute it directly out of the Tcl simulation script for the simulation which you want to visualize. Below you can see a screenshot of a nam window where the most important functions are being explained. explained.



How to start

Now we are going to write a 'template' that you can use for all of the first Tcl scripts. You can write your Tcl scripts in any text editor like joe or emacs. I suggest that you call this first example 'example1.tcl'.

First of all, you need to create a simulator object. This is done with the command

```
set ns [new Simulator]
```

Now we open a file for writing that is going to be used for the nam trace data.

```
set nf [open out.nam w]
$ns namtrace-all $nf
```

The first line opens the file 'out.nam' for writing and gives it the file handle 'nf'. In the second line we tell the simulator object that we created above to write all simulation data that is going to be relevant for nam into this file.The next step is to add a 'finish' procedure that closes the trace file and starts nam.

```
proc finish {} {
global ns nf
$ns flush-trace
close $nf
exec nam out.nam &
exit 0
}
```

You don't really have to understand all of the above code yet. It will get clearer to you once you see what the code does. The next line tells the simulator object to execute the 'finish' procedure after 5.0 seconds of simulation time.

```
$ns at 5.0 "finish"
```

You probably understand what this line does just by looking at it. ns provides you with a very simple way to schedule events with the 'at' command. The last line finally starts the simulation.

```
$ns run
```

You can actually save the file now and try to run it with 'ns example1.tcl'. You are going to get an error message like 'nam: empty trace file out.nam' though, because until now we haven't defined any objects (nodes, links, etc.) or events. You will have to use the code from this section as starting point in the other sections.

```
#Create a simulator object
set ns [new Simulator]

#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Define a 'finish' procedure
proc finish {} {
global ns nf
$ns flush-trace

#Close the trace file
close $nf

#Execute nam on the trace file
exec nam out.nam &
exit 0
}

# Insert your own code for topology creation
# and agent definitions, etc. here
#Call the finish procedure after 5 seconds simulation time
$ns at 5.0 "finish"
```

#Run the simulation
$ns run

Two nodes, one link

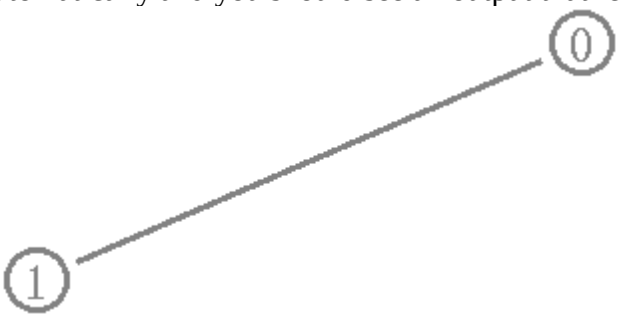
In this section we are going to define a very simple topology with two nodes that are connected by a link. The following two lines define the two nodes. (Note: You have to insert the code in this section **before** the line '$ns run', or even better, before the line '$ns at 5.0 "finish"').

set n0 [$ns node]
set n1 [$ns node]

A new node object is created with the command '$ns node'. The above code creates two nodes and assigns them to the handles 'n0' and 'n1'. The next line connects the two nodes.

$ns duplex-link $n0 $n1 1Mb 10ms DropTail

This line tells the simulator object to connect the nodes n0 and n1 with a duplex link with the bandwidth 1Megabit, a delay of 10ms and a DropTail queue.
Now you can save your file and start the script with 'ns example1.tcl'. nam will be started automatically and you should see an output that resembles the picture below.



#Create a simulator object
set ns [new Simulator]
#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf
#Define a 'finish' procedure
proc finish {} {
global ns nf
$ns flush-trace
#Close the trace file
close $nf
#Execute nam on the trace file
exec nam out.nam &
exit 0
}
#Create two nodes
set n0 [$ns node]
set n1 [$ns node]
#Create a duplex link between the nodes

$ns duplex-link $n0 $n1 1Mb 10ms DropTail
#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"
#Run the simulation
$ns run

Sending data

The next step is to send some data from node n0 to node n1. In ns, data is always being sent from one 'agent' to another. So the next step is to create an agent object that sends data from node n0, and another agent object that receives the data on node n1.

#Create a UDP agent and attach it to node n0
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0

 # Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

These lines create a UDP agent and attach it to the node n0, then attach a CBR traffic generator to the UDP agent. CBR stands for 'constant bit rate'. Line 7 and 8 should be self-explaining. The packet Size is being set to 500 bytes and a packet will be sent every 0.005 seconds (i.e. 200 packets per second). The next lines create a Null agent which acts as traffic sink and attach it to node n1.

set null0 [new Agent/Null]
$ns attach-agent $n1 $null0

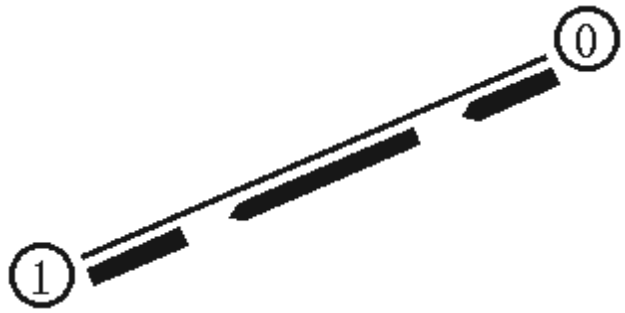
Now the two agents have to be connected with each other.

$ns connect $udp0 $null0

And now we have to tell the CBR agent when to send data and when to stop sending. Note: It's probably best to put the following lines just before the line '$ns at 5.0 "finish"'.

$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"

This code should be self-explaining again. Now you can save the file and start the simulation again. When you click on the 'play' button in the nam window, you will see that after 0.5 simulation seconds, node 0 starts sending data packets to node 1. You might want to slow nam down then with the 'Step' slider.



Now you start some experiments with nam and the Tcl script. You can click on any packet in the nam window to monitor it, and you can also click directly on the link to get some graphs with statistics.Try to change the 'packetsize_' and 'interval_' parameters in the Tcl script to see what happens.

```
#Create a simulator object
set ns [new Simulator]
#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf
#Define a 'finish' procedure
proc finish {} {
global ns nf
$ns flush-trace
#Close the trace file
close $nf
#Execute nam on the trace file
exec nam out.nam &
exit 0
}

#Create two nodes
set n0 [$ns node]
set n1 [$ns node]
#Create a duplex link between the nodes
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
#Create a UDP agent and attach it to node n0
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
# Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
#Create a Null agent (a traffic sink) and attach it to node n1
set null0 [new Agent/Null]
$ns attach-agent $n1 $null0

#Connect the traffic source with the traffic sink
$ns connect $udp0 $null0
#Schedule events for the CBR agent
```

$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"

#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"
#Run the simulation
$ns run

The topology

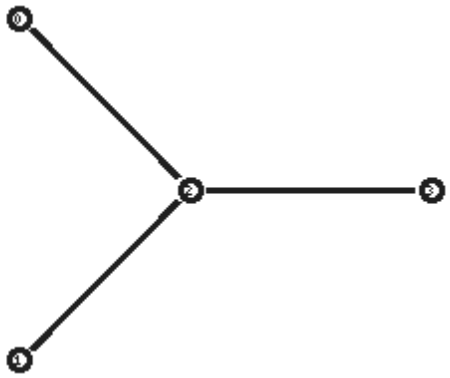
You will always have to create a simulator object, you will always have to start the simulation with the same command, and if you want to run nam automatically, you will always have to open a trace file, initialize it, and define a procedure which closes it and starts nam.

Now insert the following lines into the code to create four nodes.
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

The following piece of Tcl code creates three duplex links between the nodes.
$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n3 $n2 1Mb 10ms DropTail

You can save and start the script now. You might notice that the topology looks a bit awkward in nam. You can hit the 're-layout' button to make it look better, but it would be nice to have some more control over the layout. Add the next three lines to your Tcl script and start it again.
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right

You will probably understand what this code does when you look at the topology in the nam window now. It should look like the picture below.



The events

Now we create two UDP agents with CBR traffic sources and attach them to the nodes n0 and n1. Then we create a Null agent and attach it to node n3.

```
#Create a UDP agent and attach it to node n0
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0

# Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

#Create a UDP agent and attach it to node n1
set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1

# Create a CBR traffic source and attach it to udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1
set null0 [new Agent/Null]
$ns attach-agent $n3 $null0
```

The two CBR agents have to be connected to the Null agent.

```
$ns connect $udp0 $null0
$ns connect $udp1 $null0
```

We want the first CBR agent to start sending at 0.5 seconds and to stop at 4.5 seconds while the second CBR agent starts at 1.0 seconds and stops at 4.0 seconds.

```
$ns at 0.5 "$cbr0 start"
$ns at 1.0 "$cbr1 start"

$ns at 4.0 "$cbr1 stop"
$ns at 4.5 "$cbr0 stop"
```

When you start the script now with 'ns example2.tcl', you will notice that there is more traffic on the links from n0 to n2 and n1 to n2 than the link from n2 to n3 can carry. A simple calculation confirms this: We are sending 200 packets per second on each of the first two links and the packet size is 500 bytes. This results in a bandwidth of 0.8 megabits per second for the links from n0 to n2 and from n1 to n2. That's a total bandwidth of 1.6Mb/s, but the link between n2 and n3 only has a capacity of 1Mb/s, so obviously some packets are being discarded. But which ones? Both flows are black, so the only way to find out what is happening to the packets is to monitor them in nam by clicking on them. In the next two

sections I'm going to show you how to distinguish between different flows and how to see what is actually going on in the queue at the link from n2 to n3.

Marking flows Add the following two lines to your CBR agent definitions.

$udp0 set class_ 1

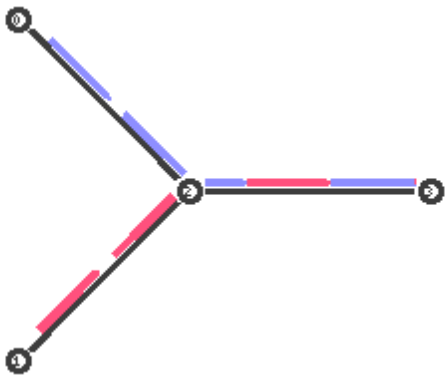
$udp1 set class_ 2

The parameter 'fid_' stands for 'flow id'.

Now add the following piece of code to your Tcl script, preferably at the beginning after the simulator object has been created, since this is a part of the simulator setup.

$ns color 1 Blue
$ns color 2 Red

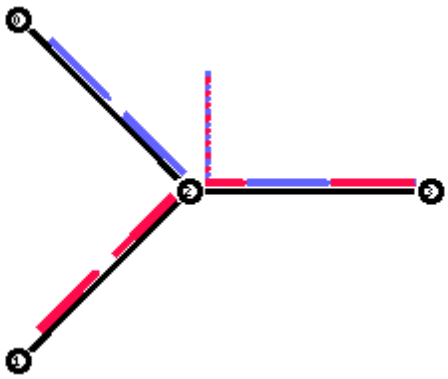
This code allows you to set different colors for each flow id.



Now you can start the script again and one flow should be blue, while the other one is red. Watch the link from node n2 to n3 for a while, and you will notice that after some time the distribution between blue and red packets isn't too fair anymore (at least that's the way it is on my system). In the next section I'll show you how you can look inside this link's queue to find out what is going on there.

Monitoring a queue You only have to add the following line to your code to monitor the queue for the link from n2 to n3.

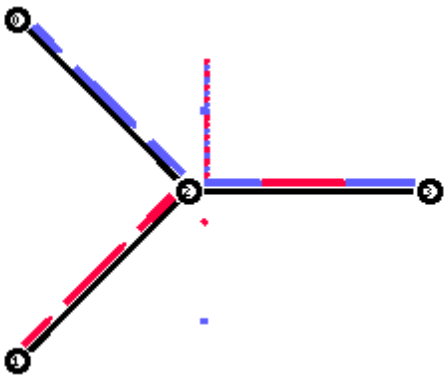
$ns duplex-link-op $n2 $n3 queuePos 0.5
Start ns again and you will see a picture similar to the one below after a few moments.



You can see the packets in the queue now, and after a while you can even see how the packets are being dropped, though (at least on my system, I guess it might be different in later or earlier releases) only blue packets are being dropped. But you can't really expect too much 'fairness' from a simple Drop Tail queue. So let's try to improve the queuing by using a SFQ (stochastic fair queuing) queue for the link from n2 to n3. Change the link definition for the link between n2 and n3 to the following line.

$ns duplex-link $n3 $n2 1Mb 10ms SFQ

The queuing should be 'fair' now. The same amount of blue and red packets should be dropped.



Simple Programs Using NS2

9.1) Positioning the Nodes

```
set ns [new Simulator]
set nf [open out.nam w]
$ns namtrace-all $nf

proc finish {} {
        global ns nf
        $ns flush-trace
        close $nf
        exec nam out.nam &
        exit 0
    }

set n0 [$ns node]
set n1 [$ns node]
```

```
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

$ns duplex-link $n0 $n1 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 2Mb 10ms DropTail
$ns duplex-link $n3 $n4 2Mb 10ms DropTail
$ns duplex-link $n4 $n5 2Mb 10ms DropTail
$ns duplex-link $n5 $n0 2Mb 10ms DropTail

$ns duplex-link-op $n0 $n1 orient right-up
$ns duplex-link-op $n1 $n2 orient right
$ns duplex-link-op $n2 $n3 orient right-down
$ns duplex-link-op $n2 $n3 orient right-down
$ns duplex-link-op $n0 $n5 orient right-down
$ns duplex-link-op $n5 $n4 orient right
$ns duplex-link-op $n4 $n3 orient right-up

$ns at 5.0 "finish"
$ns run
```

9.2) Setting wired simulation environment

```
set ns [new Simulator]
set nf [open out.nam w]
$ns namtrace-all $nf

proc finish {} {
	global ns nf
	$ns flush-trace
	close $nf
	exec nam out.nam &
	exit 0

}

set n0 [$ns node]
set n1 [$ns node]
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns at 5.0 "finish"
$ns run
```

9.3) TCP connection For wired Network Simulation

```
set ns [new Simulator]
set nf [open out.nam w]
```

```
$ns namtrace-all $nfproc
finish {}
{
global ns nf
$ns flush-trace close
$nf exec nam out.nam &
exit 0
}
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

$ns duplex-link $n0 $n1 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 2Mb 10ms DropTail
$ns duplex-link $n3 $n4 2Mb 10ms DropTail
$ns duplex-link $n4 $n5 2Mb 10ms DropTail
$ns duplex-link $n5 $n0 2Mb 10ms DropTail

$ns duplex-link-op $n0 $n1 orient right-up
$ns duplex-link-op $n1 $n2 orient right
$ns duplex-link-op $n2 $n3 orient right-down
$ns duplex-link-op $n2 $n3 orient right-down
$ns duplex-link-op $n0 $n5 orient right-down
$ns duplex-link-op $n5 $n4 orient right
$ns duplex-link-op $n4 $n3 orient right-up

set tcp [new Agent/TCP]
$tcp set class_ 2
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP
$ns at 0.1 "$ftp start"
$ns at 2.9 "$ftp stop"
$ns at 3.0 "finish"
$ns run
```

9.4) Fixed Colored wireless nodes

```
# Define options
set val(chan) Channel/WirelessChannel ;# channel type
```

```
set val(prop) Propagation/TwoRayGround ;# radio-propagation model
set val(netif) Phy/WirelessPhy ;# network interface type
set val(mac) Mac/802_11 ;# MAC type
set val(ifq) Queue/DropTail/PriQueue ;# interface queue type
set val(ll) LL ;# link layer type
set val(ant) Antenna/OmniAntenna ;# antenna model
set val(ifqlen) 50 ;# max packet in ifq
set val(nn) 8 ;# number of mobilenodes
set val(rp) AODV ;# routing protocol
set val(x) 500 ;# X dimension of topography
set val(y) 400 ;# Y dimension of topography
set val(stop) 10 ;# time of simulation end

#-------Event scheduler object creation--------#

set ns              [new Simulator]

#Creating trace file and nam file.
set tracefd       [open wireless2.tr w]
set namtrace      [open wireless2.nam w]

$ns trace-all $tracefd
$ns namtrace-all-wireless $namtrace $val(x) $val(y)

# set up topography object
set topo [new Topography]

$topo load_flatgrid $val(x) $val(y)

set god_ [create-god $val(nn)]

# configure the nodes
        $ns node-config -adhocRouting $val(rp) \
                        -llType $val(ll) \
                        -macType $val(mac) \
                        -ifqType $val(ifq) \
                        -ifqLen $val(ifqlen) \
                        -antType $val(ant) \
                        -propType $val(prop) \
                        -phyType $val(netif) \
                        -channelType $val(chan) \
                        -topoInstance $topo \
                        -agentTrace ON \
                        -routerTrace ON \
                        -macTrace OFF \
                        -movementTrace ON

# Creating node objects..
for {set i 0} {$i < $val(nn) } { incr i } {
            set node_($i) [$ns node]
```

```
    }
    for {set i 0} {$i < $val(nn)  } {incr i } {
        $node_($i) color cyan
        $ns at 0.0 "$node_($i) color cyan"
    }

# Provide initial location of mobilenodes
$node_(0) set X_ 5.0
$node_(0) set Y_ 30.0
$node_(0) set Z_ 0.0

$node_(1) set X_ 50.0
$node_(1) set Y_ 25.0
$node_(1) set Z_ 0.0

$node_(2) set X_ 200.0
$node_(2) set Y_ 90.0
$node_(2) set Z_ 0.0

$node_(3) set X_ 350.0
$node_(3) set Y_ 160.0
$node_(3) set Z_ 0.0

$node_(4) set X_ 100.0
$node_(4) set Y_ 250.0
$node_(4) set Z_ 0.0

$node_(5) set X_ 300.0
$node_(5) set Y_ 100.0
$node_(5) set Z_ 0.0

$node_(6) set X_ 400.0
$node_(6) set Y_ 350.0
$node_(6) set Z_ 0.0

$node_(7) set X_ 3s50.0
$node_(7) set Y_ 470.0
$node_(7) set Z_ 0.0

# Define node initial position in nam
for {set i 0} {$i < $val(nn)} { incr i } {
# 30 defines the node size for nam
$ns initial_node_pos $node_($i) 30
}

# Telling nodes when the simulation ends
for {set i 0} {$i < $val(nn) } { incr i } {
   $ns at $val(stop) "$node_($i) reset";
}
```

```
# ending nam and the simulation
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "stop"
$ns at 10.01 "puts \"end simulation\" ; $ns halt"
proc stop {} {
   global ns tracefd namtrace
   $ns flush-trace
   close $tracefd
   close $namtrace
exec nam wireless2.nam &
}

$ns run
```

Observation	
Record	
Total	
Staff Signature	