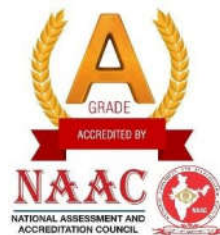




Bharath
INSTITUTE OF HIGHER EDUCATION AND RESEARCH
(Declared as Deemed - to - be - University under section 3 of UGC Act 1956)



BHARATH INSTITUTE OF SCIENCE & TECHNOLOGY

173, Agaram Road, Selaiyur, Chennai-600073. Tamil Nadu, India.

SCHOOL OF COMPUTING

*Department of **Computer Science & Engineering***

BACHELOR OF TECHNOLOGY

COURSE CODE: U20CSCJ07

NETWORK AND COMMUNICATION

LABORATORY RECORD

Name of the Student:

Batch: 2020-2024

Year: III

Term/Semester: V

Section:

Register No:

DEC 2022



Bharath

INSTITUTE OF HIGHER EDUCATION AND RESEARCH
(Declared as Deemed - to - be - University under section 3 of UGC Act 1956)



BHARATH INSTITUTE OF SCIENCE & TECHNOLOGY

173, Agaram Road, Selaiyur, Chennai-600073, Tamil Nadu, India.

Name: _____

Programme: _____ Branch: _____

Year : _____ Semester : _____

Register No:

Certified that this is the bonafide record of work done by the above student in the
..... laboratory during the
..... Semester in the Academic Year 2022-2023.

Faculty Incharge

Head of Department

Submitted for the practical Examination held on

Internal Examiner

External Examiner

INDEX

Exp No	DATE OF COMPLETION	NAME OF THE PROGRAM	PAGE NO	SIGNATURE OF THE FACULTY
1		Create a socket (TCP) between two computers and enable file transfer between them.		
2		Write a socket program to demonstrate Echo/Ping/Talk commands.		
3		Write a program to develop a simple Chat Using TCP application.		
4		Write a program to develop a simple Chat Using UDP application.		
5		Implementation of Stop and Wait Protocol and Sliding Window Protocol.		
6		Implementation of DNS, SNMP and File Transfer application using TCP and UDP Sockets.		
7		Create a socket for HTTP for web page upload and download.		
8		Develop a program to display the client's address at the server end.		
9		Study of Network simulator (NS) and Simulation of Congestion Control Algorithms using NS		
10		Perform a case study about the different routing algorithms to select the network path with its optimum and economical during data transfer. i. Link State routing ii. Flooding iii. Distance vector		

EX.NO:1	Create a socket (TCP) between two computers and enable file transfer between them.
DATE:	

AIM

To Perform File Transfer in Client & Server Using TCP/IP.

ALGORITHM

CLIENT SIDE

1. Start.
2. Establish a connection between the Client and Server.
3. Socket ss=new Socket(InetAddress.getLocalHost(),2000);
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 client either get file from the server or send
6. 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

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

class Pings
{
public static void main(String args[]) throws IOException
{
ServerSocket s = new ServerSocket(2000);
while(true)
{
Socket c = s.accept();
InputStream in = c.getInputStream();
InputStreamReader inr = new InputStreamReader(in);
BufferedReader br = new BufferedReader(inr);
String str = br.readLine();
System.out.println("Ping command received from : "+c.getInetAddress() +" with string "+str);
PrintStream ps = new PrintStream(c.getOutputStream());
ps.println(str);
if(str.equals("exit"))
{
break;
}
}
}
```

//CLIENT

```
import java.io.*;
import java.net.*;
public class Pingc
{
```

```

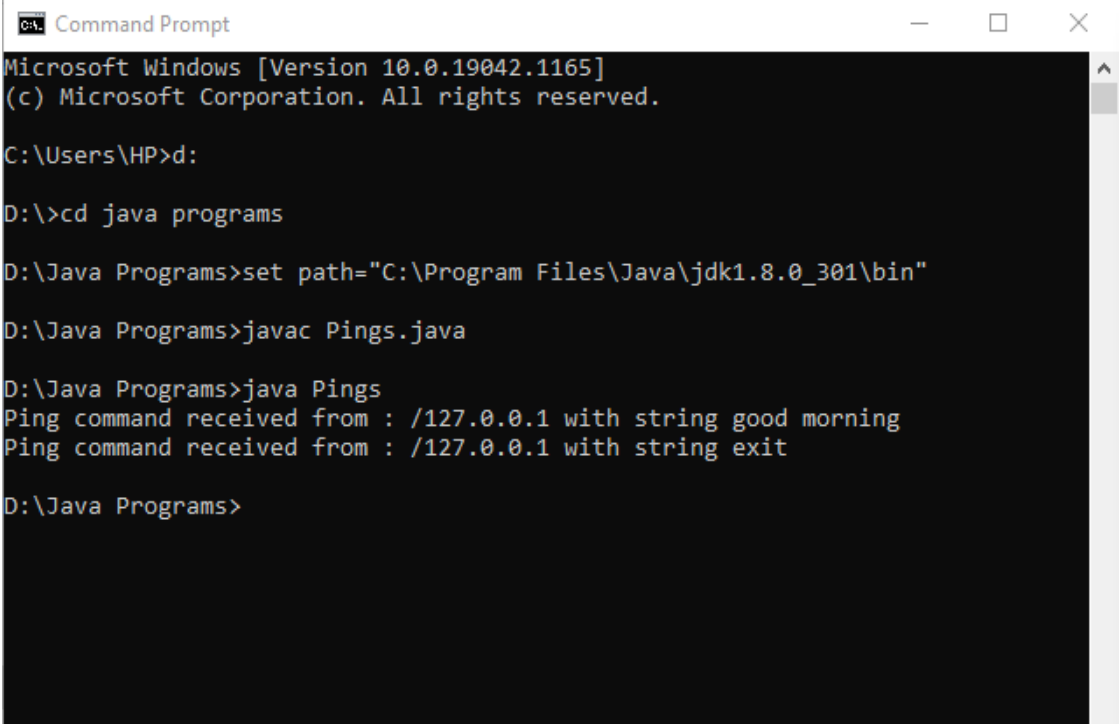
public static void main(String args[]) throws IOException
{
    long t1, t2;
    while(true)
    {
        try
        {
            Socket soc = new Socket("localhost",2000);
            InputStreamReader isr = new InputStreamReader(System.in);
            BufferedReader br = new BufferedReader(isr);
            System.out.println("Type a string to ping : ");
            String str = br.readLine();
            OutputStream os = soc.getOutputStream();
            PrintWriter pw = new PrintWriter(os,true);
            InputStream in = soc.getInputStream();
            InputStreamReader inr = new InputStreamReader(in);
            BufferedReader br1 = new BufferedReader(inr);
            t1 = System.currentTimeMillis();
            pw.println(str);
            String str1 = br1.readLine();
            t2 = System.currentTimeMillis();
            System.out.println("Pinging "+soc.getInetAddress()+" with string "+str );
            System.out.println("Reply from "+soc.getInetAddress() +" String "+str1+" Length: "+str1.length());
            System.out.println("Sent : "+str.length()+" Received : "+str1.length()+" Lost : "+(str.length()-
            str1.length()));
            System.out.println("Approx. Time in Milliseconds = "+(t2-t1));
            if(str.equals("exit"))
            {
                break;
            }
        }
        catch(Exception e)
        {

```

```
System.out.println("Error : "+e.getMessage());  
}  
}  
}  
}
```

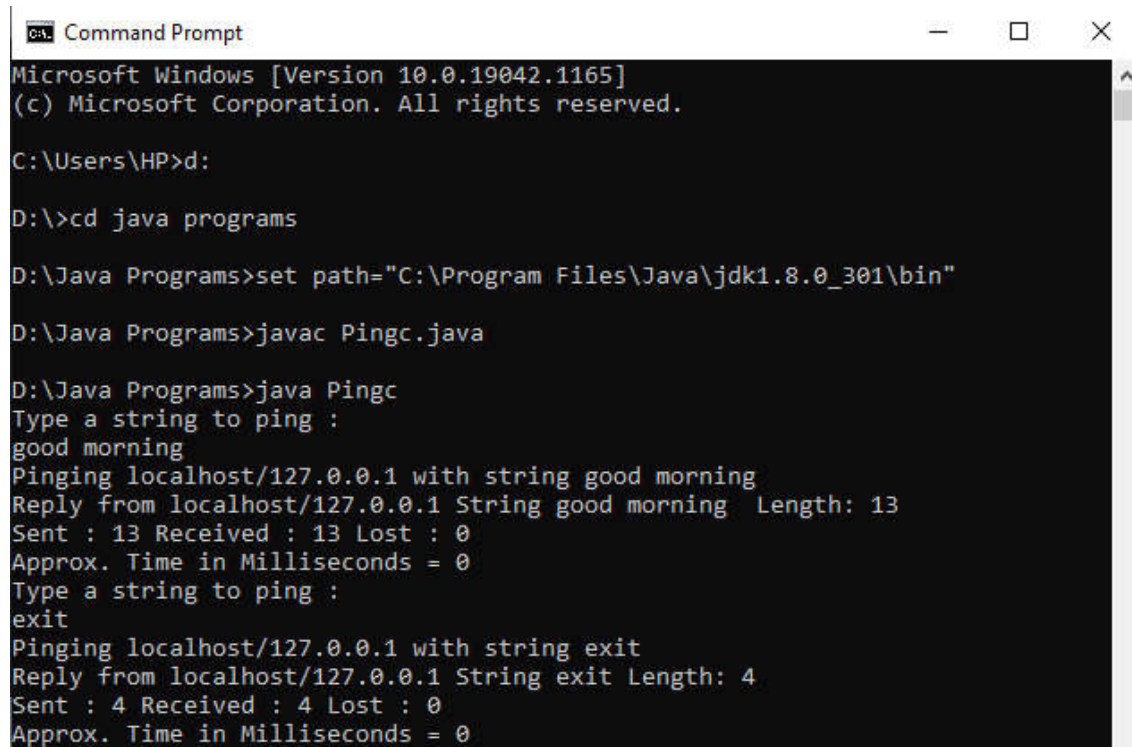
OUTPUT

SERVER



```
Command Prompt  
Microsoft Windows [Version 10.0.19042.1165]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Users\HP>d:  
  
D:\>cd java programs  
  
D:\Java Programs>set path="C:\Program Files\Java\jdk1.8.0_301\bin"  
  
D:\Java Programs>javac Pings.java  
  
D:\Java Programs>java Pings  
Ping command received from : /127.0.0.1 with string good morning  
Ping command received from : /127.0.0.1 with string exit  
  
D:\Java Programs>
```

CLIENT



```
ca. Command Prompt
Microsoft Windows [Version 10.0.19042.1165]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP>d:

D:\>cd java programs

D:\Java Programs>set path="C:\Program Files\Java\jdk1.8.0_301\bin"

D:\Java Programs>javac Pingc.java

D:\Java Programs>java Pingc
Type a string to ping :
good morning
Pinging localhost/127.0.0.1 with string good morning
Reply from localhost/127.0.0.1 String good morning Length: 13
Sent : 13 Received : 13 Lost : 0
Approx. Time in Milliseconds = 0
Type a string to ping :
exit
Pinging localhost/127.0.0.1 with string exit
Reply from localhost/127.0.0.1 String exit Length: 4
Sent : 4 Received : 4 Lost : 0
Approx. Time in Milliseconds = 0
```

RESULT

Thus the java program for file transfer Operation is done & executed successfully.

EX.NO:2	Create a socket Program for Echo/Ping/Talk commands.
DATE:	

AIM

To write a socket program for Echo /Ping /Talk commands.

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

//ECHO SERVER

```
import java.io.*;
import java.net.*;
import java.lang.*;
public class eserver
{
public static void main(String args[])throws IOException
{
ServerSocket s=null;
String line;
DataInputStream is;
PrintStream ps;
Socket c=null;
try
{
s=new ServerSocket(8080);
}
catch(IOException e)
{
System.out.println(e);
}
try
{
c=s.accept();
is=new DataInputStream(c.getInputStream());
ps=new PrintStream(c.getOutputStream());
while(true)
{
line=is.readLine();
System.out.println("msg received and sent back to client");
ps.println(line);
```

```

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

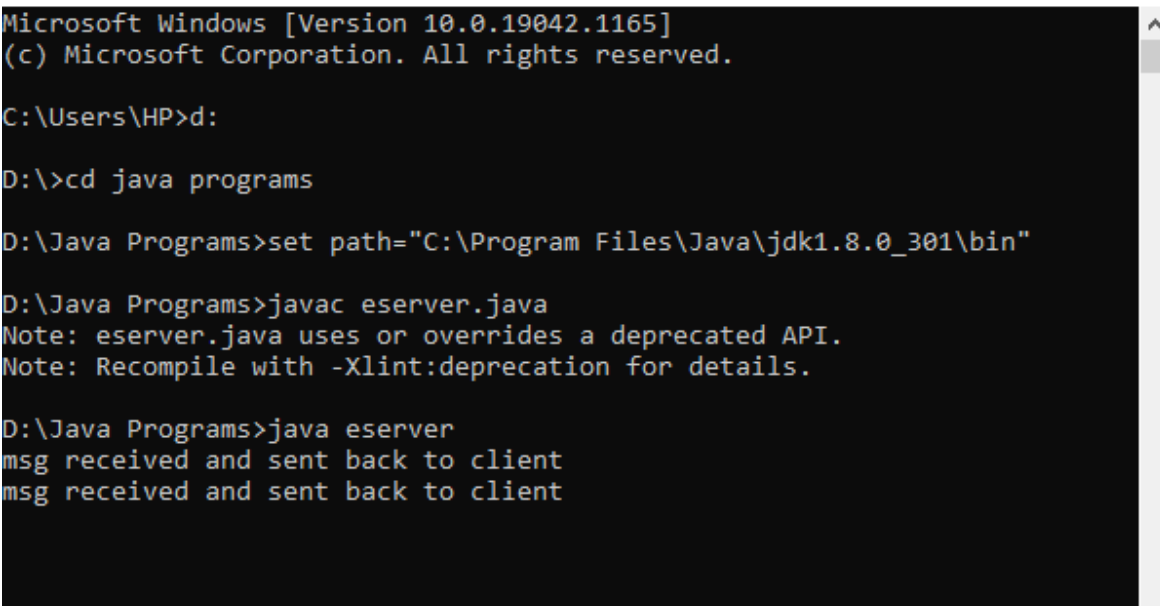
//ECHO CLIENT
import java.io.*;
import java.net.*;
public class eclient
{
    public static void main(String args[])
    {
        Socket c=null;
        String line;
        DataInputStream is,is1;
        PrintStream os;
        try
        {
            c=new Socket("localhost",8080);
        }
        catch(IOException e)
        {
            System.out.println(e);
        }
        try
        {
            os=new PrintStream(c.getOutputStream());
            is=new DataInputStream(System.in);
            is1=new DataInputStream(c.getInputStream());
            do

```

```
{
System.out.println("client");
line=is.readLine();
os.println(line);
if(!line.equals("exit"))
System.out.println("server:"+is1.readLine());
}while(!line.equals("exit"));
}
catch(IOException e)
{
System.out.println("socket closed");
}
}
}
```

OUTPUT

SERVER



```
Microsoft Windows [Version 10.0.19042.1165]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP>d:

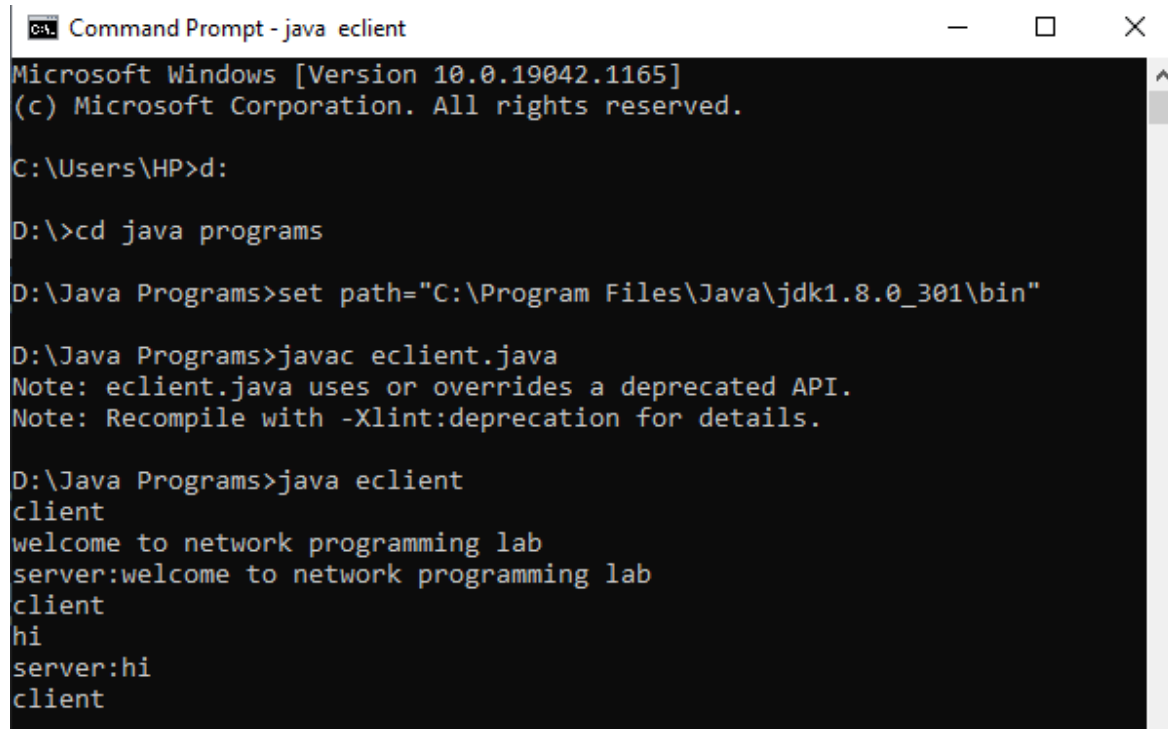
D:\>cd java programs

D:\Java Programs>set path="C:\Program Files\Java\jdk1.8.0_301\bin"

D:\Java Programs>javac eserver.java
Note: eserver.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

D:\Java Programs>java eserver
msg received and sent back to client
msg received and sent back to client
```

CLIENT



```
Command Prompt - java eclient
Microsoft Windows [Version 10.0.19042.1165]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP>d:

D:\>cd java programs

D:\Java Programs>set path="C:\Program Files\Java\jdk1.8.0_301\bin"

D:\Java Programs>javac eclient.java
Note: eclient.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

D:\Java Programs>java eclient
client
welcome to network programming lab
server:welcome to network programming lab
client
hi
server:hi
client
```

RESULT

Thus the program socket program for Echo /Ping /Talk commands was written & executed successfully.

EX.NO:3	Develop a simple Chat Program Using TCP Application.
DATE:	

AIM

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

ALGORITHM

CLIENT

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 accept 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

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
7. The server communicate the client to send the end of the message.
8. Stop the program.

PROGRAM

//SERVER

```
import java.io.*;
import java.net.*;
public class Server
{
    public static void main(String[] args) throws Exception
    {
        ServerSocket sersock = new ServerSocket(3000);
        System.out.println("Server ready for chatting");
        Socket sock = sersock.accept( );
        BufferedReader keyRead = new BufferedReader(new InputStreamReader(System.in)); // reading from
        keyboard (keyRead object)
        OutputStream ostream = sock.getOutputStream(); // sending to client (pwrite object)
        PrintWriter pwrite = new PrintWriter(ostream, true);
        InputStream istream = sock.getInputStream();// receiving from server ( receiveRead object)
        BufferedReader receiveRead = new BufferedReader(new InputStreamReader(istream));
        String receiveMessage, sendMessage;
        while(true)
        {
            if((receiveMessage = receiveRead.readLine()) != null)
            {
                System.out.println(receiveMessage);
            }
            sendMessage = keyRead.readLine();
            pwrite.println(sendMessage);
            pwrite.flush();
        }
    }
}
```

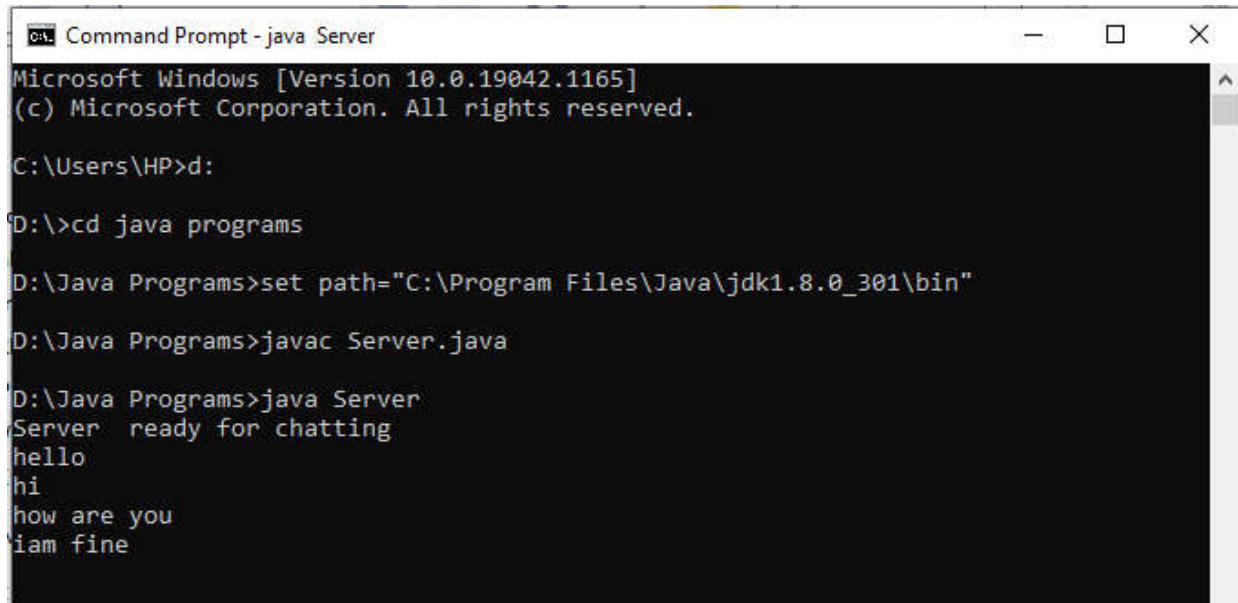
```

//CLIENT
import java.io.*;
import java.net.*;
public class Client
{
    public static void main(String[] args) throws Exception
    {
        Socket sock = new Socket("127.0.0.1", 3000);
        // reading from keyboard (keyRead object)
        BufferedReader keyRead = new BufferedReader(new InputStreamReader(System.in));
        // sending to client (pwrite object)
        OutputStream ostream = sock.getOutputStream();
        PrintWriter pwrite = new PrintWriter(ostream, true);
        // receiving from server ( receiveRead object)
        InputStream istream = sock.getInputStream();
        BufferedReader receiveRead = new BufferedReader(new InputStreamReader(istream));
        System.out.println("Start the chitchat, type and press Enter key");
        String receiveMessage, sendMessage;
        while(true)
        {
            sendMessage = keyRead.readLine(); // keyboard reading
            pwrite.println(sendMessage);    // sending to server
            pwrite.flush();                // flush the data
            if((receiveMessage = receiveRead.readLine()) != null) //receive from server
            {
                System.out.println(receiveMessage); // displaying at DOS prompt
            }
        }
    }
}

```


OUTPUT:

SERVER



```
Command Prompt - java Server
Microsoft Windows [Version 10.0.19042.1165]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP>d:

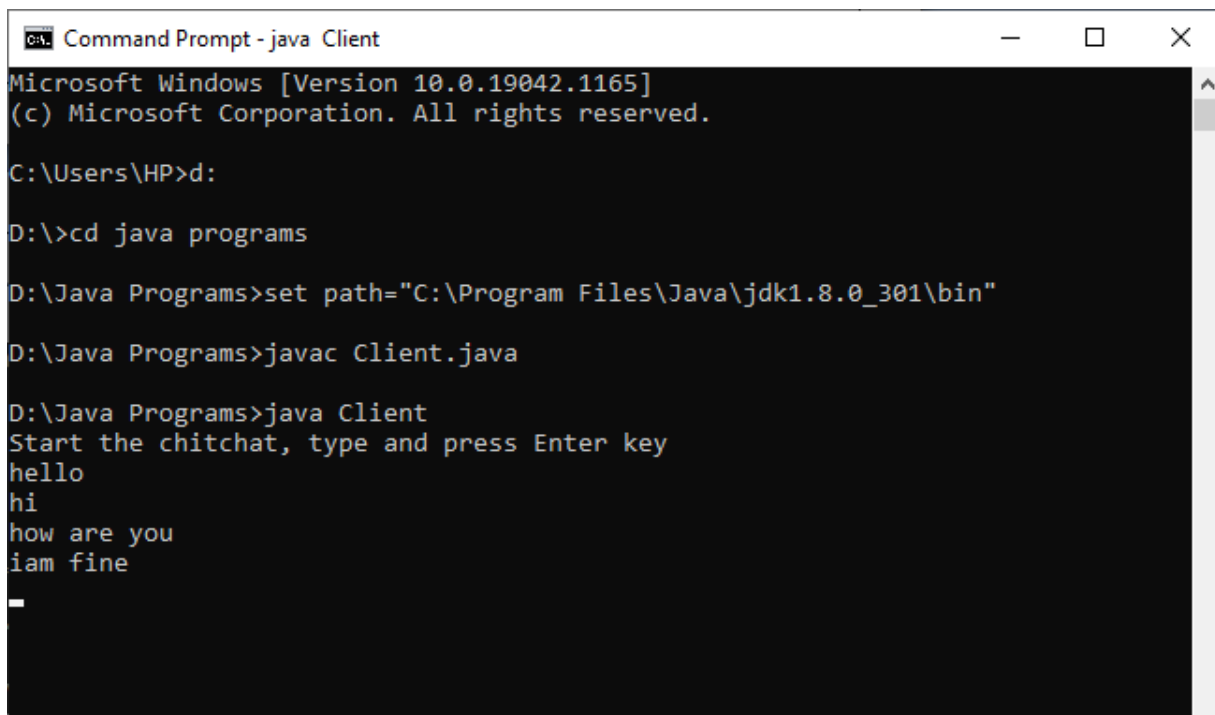
D:\>cd java programs

D:\Java Programs>set path="C:\Program Files\Java\jdk1.8.0_301\bin"

D:\Java Programs>javac Server.java

D:\Java Programs>java Server
Server ready for chatting
hello
hi
how are you
iam fine
```

CLIENT



```
Command Prompt - java Client
Microsoft Windows [Version 10.0.19042.1165]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP>d:

D:\>cd java programs

D:\Java Programs>set path="C:\Program Files\Java\jdk1.8.0_301\bin"

D:\Java Programs>javac Client.java

D:\Java Programs>java Client
Start the chitchat, type and press Enter key
hello
hi
how are you
iam fine
_
```

RESULT

Thus the Java program for simple chat using TCP / IP application was written and executed successfully.

EX.NO:4	Develop a simple Chat Program Using TCP Application.
DATE:	

AIM

To write a program to implement simple client-server application using UDP.

ALGORITHM

CLIENT SIDE

1. Create a datagram socket with server's IP address.
2. Create datagram packets with data, data length and the port address.
3. Send the datagram packets to server through datagram sockets
4. Receive the datagram packets from server through datagram sockets
5. Close the socket.

SERVER SIDE

1. Create a datagram socket with port address.
2. Create datagram packets with data, data length and the port address.
3. Send the datagram packets to client through datagram sockets
4. Receive the datagram packets from client through datagram sockets
5. Close the socket.

PROGRAM

//SERVER

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

class UDPserver
{
    public static int clientport = 8040,serverport = 8050;
    public static void main(String args[]) throws Exception
    {
        DatagramSocket SrvSoc = new DatagramSocket(clientport);
        byte[] SData = new byte[1024];
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Server Ready");
        while (true)
        {
            byte[] RData = new byte[1024];
            DatagramPacket RPack = new DatagramPacket(RData,RData.length);
            SrvSoc.receive(RPack);
            String Text = new String(RPack.getData());
            if (Text.trim().length() == 0)
                break;
            System.out.println("\nFrom Client <<< " + Text );
            System.out.print("Msg to Client : " );
            String srvmsg = br.readLine();
            InetAddress IPAddr = RPack.getAddress();
            SData = srvmsg.getBytes();
            DatagramPacket SPack = new DatagramPacket(SData,SData.length,IPAddr, serverport);
            SrvSoc.send(SPack);
        }
        System.out.println("\nClient Quits\n");
        SrvSoc.close();
    }
}
```

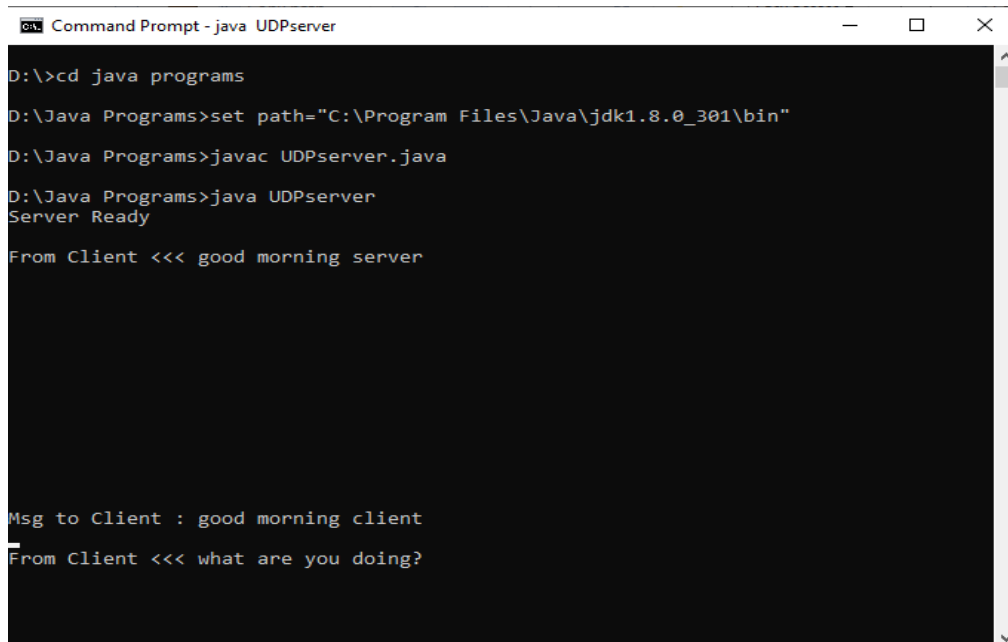
```

//CLIENT
import java .io.*;
import java.net.*;
class UDPclient
{
public static DatagramSocket ds;
public static int clientport=789,serverport=790;
public static void main(String args[])throws Exception
{
byte buffer[]=new byte[1024];
ds=new DatagramSocket(serverport);
BufferedReader dis=new BufferedReader(new InputStreamReader(System.in));
System.out.println("server waiting");
InetAddress ia=InetAddress.getByName("10.0.200.36");
while(true)
{
System.out.println("Client:");
String str=dis.readLine();
if(str.equals("end")) break;
buffer=str.getBytes();
ds.send(new DatagramPacket(buffer,str.length(),ia,clientport));
DatagramPacket p=new DatagramPacket(buffer,buffer.length);
ds.receive(p);String psx=new String(p.getData(),0,p.getLength());
System.out.println("Server:" + psx);
}
}
}

```

OUTPUT

SERVER

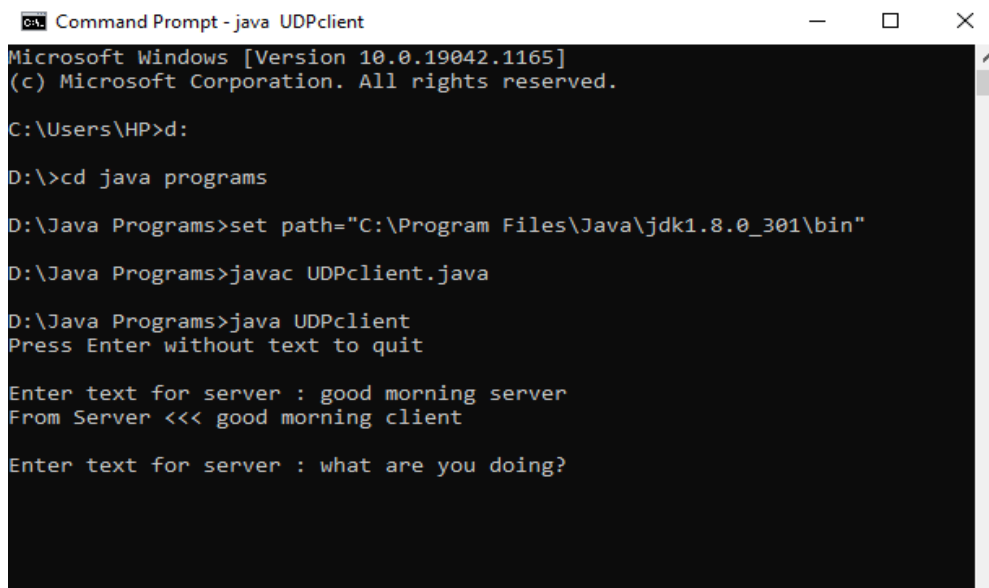


```
Command Prompt - java UDPserver

D:\>cd java programs
D:\Java Programs>set path="C:\Program Files\Java\jdk1.8.0_301\bin"
D:\Java Programs>javac UDPserver.java
D:\Java Programs>java UDPserver
Server Ready
From Client <<< good morning server

Msg to Client : good morning client
From Client <<< what are you doing?
```

CLIENT



```
Command Prompt - java UDPclient

Microsoft Windows [Version 10.0.19042.1165]
(c) Microsoft Corporation. All rights reserved.
C:\Users\HP>d:
D:\>cd java programs
D:\Java Programs>set path="C:\Program Files\Java\jdk1.8.0_301\bin"
D:\Java Programs>javac UDPclient.java
D:\Java Programs>java UDPclient
Press Enter without text to quit
Enter text for server : good morning server
From Server <<< good morning client
Enter text for server : what are you doing?
```

RESULT

Thus the Java program for simple chat using UDP application was written and executed successfully.

EX.NO:5	Implementation of Stop and Wait Protocol and Sliding Window Protocol.
DATE:	

AIM

To provide a reliable data transfer between two nodes over an unreliable network using the Stop and Wait Protocol.

ALGORITHM

Sender:

1. Establish socket connection to the receiver.
2. After successful connection with the receiver, acknowledgement will be received
3. Get the number of frames need to be sent by the sender.
4. Each frame will have unique sequence number and sent in order
5. The sender will wait for the acknowledgement for every successful transmission of each frame from receiver side.
6. In case of unsuccessful transmission, the sender will not receive any acknowledgement and again the particular frame will be re-transmitted.
7. End the program

Receiver

1. Establish the sender's socket connection.
2. After successful connection, send acknowledgement to the sender
3. With each frame received with their unique sequence number, acknowledgement message will be sent.
4. In case of faulty frame structure, the receiver will not send any response and it waits for the sender to retransmit the message again.
5. End the program

PROGRAM

//SERVER

```
import java.net.*;
import java.io.*;
import java.rmi.*;
public class swserver
{
public static void main(String a[])throws Exception
{
ServerSocket ser=new ServerSocket(10);
Socket s=ser.accept();
DataInputStream in=new DataInputStream(System.in);
DataInputStream in1=new DataInputStream(s.getInputStream());
String sbuff[]=new String[8];
PrintStream p;
int sptr=0,sws=8,nf,ano,i;
String ch;
do
{
p=new PrintStream(s.getOutputStream());
System.out.print("Enter the no. of frames : ");
nf=Integer.parseInt(in.readLine());
p.println(nf);
if(nf<=sws-1)
{
System.out.println("Enter "+nf+" Messages to be send\n");
for(i=1;i<=nf;i++)
{
sbuff[sptr]=in.readLine();
p.println(sbuff[sptr]);
sptr=++sptr%8;
}
}
```

```

sws-=nf;
System.out.print("Acknowledgment received");
ano=Integer.parseInt(in1.readLine());
System.out.println(" for "+ano+" frames");
sws+=nf;
}
else
{
System.out.println("The no. of frames exceeds window size");
break;
}
System.out.print("\nDo you wants to send some more frames : ");
ch=in.readLine();
p.println(ch);
}
while(ch.equals("yes"));
s.close();
}
}

```

//CLIENT

```

import java.net.*;
import java.io.*;
class swclient
{
public static void main(String a[])throws Exception
{
Socket s=new Socket(InetAddress.getLocalHost(),10);
DataInputStream in=new DataInputStream(s.getInputStream());
PrintStream p=new PrintStream(s.getOutputStream());
int i=0,rptr=-1,nf,rws=8;
String rbuf[]=new String[8];
String ch;

```



```

System.out.println();
do
{
nf=Integer.parseInt(in.readLine());
if(nf<=rws-1)
{
for(i=1;i<=nf;i++)
{
rptr=++rptr%8;
rbuf[rptr]=in.readLine();
System.out.println("The received Frame " +rptr+" is : "+rbuf[rptr]);
}
rws-=nf;
System.out.println("\nAcknowledgment sent\n");
p.println(rptr+1);
rws+=nf;
}
else
break;
ch=in.readLine();
}
while(ch.equals("yes"));
}
}

```

OUTPUT

SERVER

```
D:\Java Programs>javac swserver.java
Note: swserver.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

D:\Java Programs>java swserver
Enter the no. of frames : 5
Enter 5 Messages to be send

hello
good morning
welocme to network lab
tcp is connection oriented
udp is connectionless
Acknowledgment received for 5 frames

Do you wants to send some more frames : N
```

CLIENT

```
D:\Java Programs>javac swclient.java
Note: swclient.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

D:\Java Programs>java swclient

The received Frame 0 is : hello
The received Frame 1 is : good morning
The received Frame 2 is : welocme to network lab
The received Frame 3 is : tcp is connection oriented
The received Frame 4 is : udp is connectionless

Acknowledgment sent
```

RESULT:

Thus the java program for stop and wait and sliding window protocol was written & executed successfully.

EX.NO:6	Implementation of DNS, SNMP and File Transfer application using TCP and UDP Sockets
DATE:	

A) DNS**AIM**

To write a java program for DNS application program

ALGORITHM**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 a connection, receive the request to send the IP address for DNS .
5. send the IP address of the corresponding DNS
6. Close the socket.
7. End the program.

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 ,request for IP address of the domain name system.
4. Receive and print the IP address of the DNS.
5. Close the socket.
6. End the program.

PROGRAM

//DNS SERVER

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

public class udpdnsserver
{
    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 = {"yahoo.com", "gmail.com","cricinfo.com", "facebook.com"};
        String[] ip = {"68.180.206.184", "209.85.148.19","80.168.92.140", "69.63.189.16"};
        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();
}
}
}

```

//DNS CLIENT

```

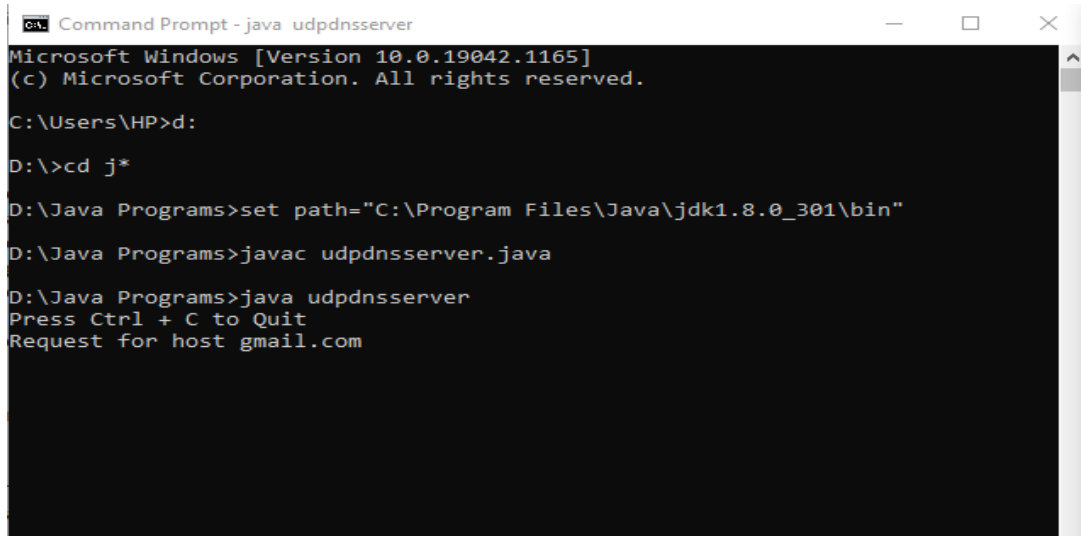
import java.io.*;
import java.net.*;
public class udpdnsclient
{
public static void main(String args[])throws IOException
{
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
DatagramSocket clientsocket = new DatagramSocket();
    InetAddress ipaddress;
if (args.length == 0)
ipaddress =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();
}
}
```

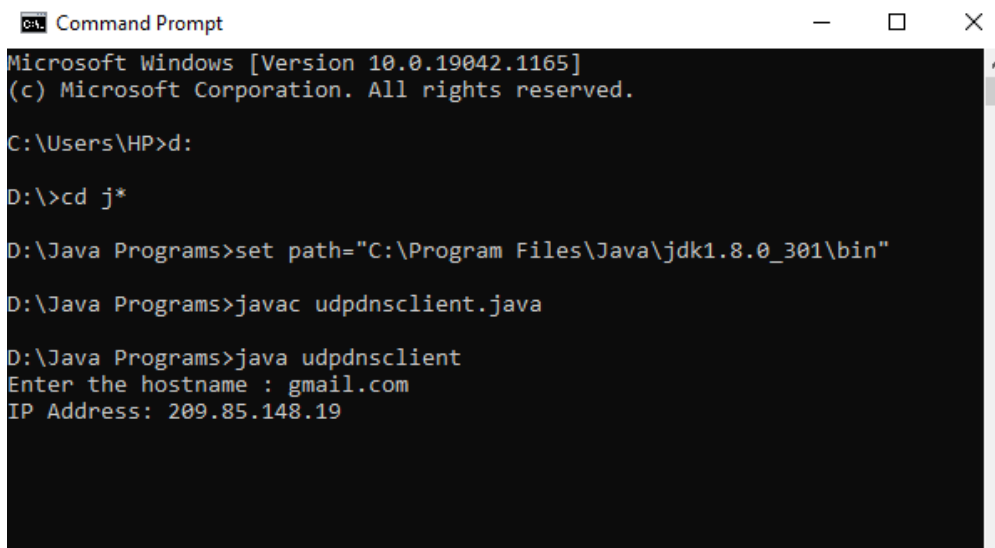
OUTPUT

SERVER



```
Command Prompt - java udpdnserver
Microsoft Windows [Version 10.0.19042.1165]
(c) Microsoft Corporation. All rights reserved.
C:\Users\HP>d:
D:\>cd j*
D:\Java Programs>set path="C:\Program Files\Java\jdk1.8.0_301\bin"
D:\Java Programs>javac udpdnserver.java
D:\Java Programs>java udpdnserver
Press Ctrl + C to Quit
Request for host gmail.com
```

CLIENT



```
Command Prompt
Microsoft Windows [Version 10.0.19042.1165]
(c) Microsoft Corporation. All rights reserved.
C:\Users\HP>d:
D:\>cd j*
D:\Java Programs>set path="C:\Program Files\Java\jdk1.8.0_301\bin"
D:\Java Programs>javac udpdnclient.java
D:\Java Programs>java udpdnclient
Enter the hostname : gmail.com
IP Address: 209.85.148.19
```

B) File Transfer 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 request for file name to be transfer
5. send the file to be print on the client
6. Close the socket.
7. End the program.

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 a connection ,request for file name to be transfer .
4. Trasfer the file and print it on client screen.
- 5.Again it will request for the file to be transfer on server side.
6. Transfer the file and print it on server screen.
5. Close the socket.
6. End the program.

//SERVER

```
import java.io.*;
import java.net.*;
import java.util.*;
class Serverfile
{
    public static void main(String args[])
    {
        try
        {
            ServerSocket obj=new ServerSocket(1300);
            while(true)
            {
                Socket obj1=obj.accept();
                DataInputStream din=new DataInputStream(obj1.getInputStream());
                DataOutputStream dout=new DataOutputStream(obj1.getOutputStream());
                String str=din.readLine();
                FileReader f=new FileReader(str);
                BufferedReader b=new BufferedReader(f);
                String s;
                while((s=b.readLine())!=null)
                {
                    System.out.println(s);
                    dout.writeBytes(s+"\n");
                }
                f.close();
                dout.writeBytes("-1\n");
            }
        }
        catch(Exception e)
        {
            System.out.println(e);}
    }
}
```

//CLIENT

```
import java.io.*;
import java.net.*;
import java.util.*;
class Clientfile
{
    public static void main(String args[])
```

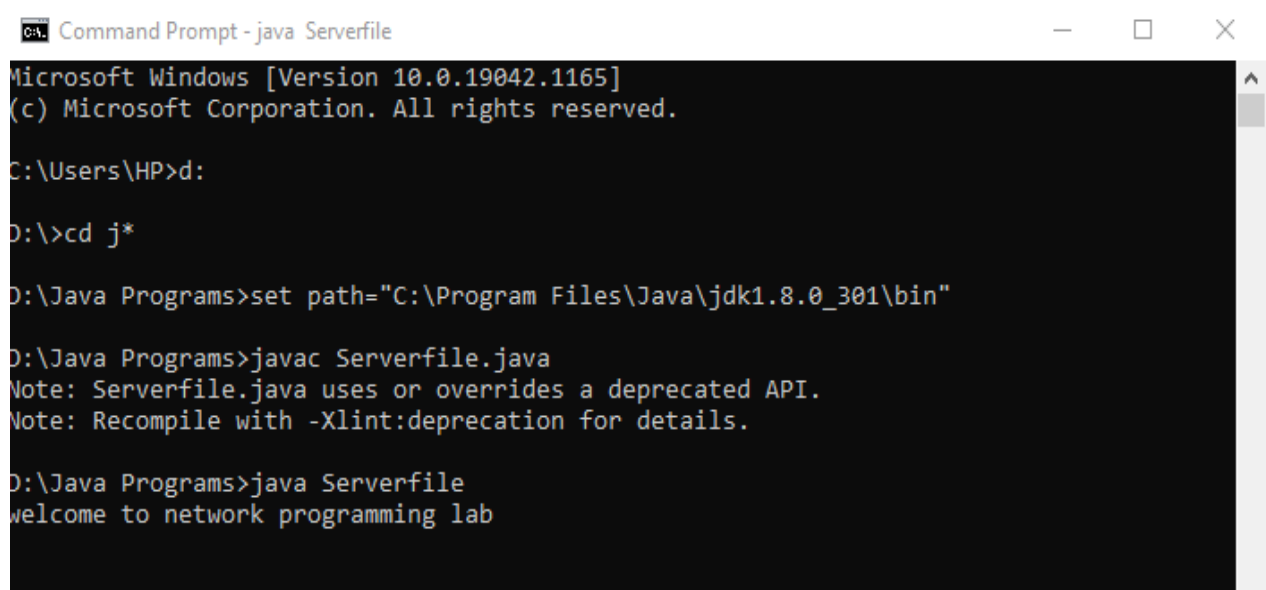


```

{
try
{
BufferedReader in=new BufferedReader(new InputStreamReader(System.in));
Socket clsct=new Socket("127.0.0.1",1300);
DataInputStream din=new DataInputStream(clsct.getInputStream());
DataOutputStream dout=new DataOutputStream(clsct.getOutputStream());
System.out.println("Enter the file name:");
String str=in.readLine();
dout.writeBytes(str+"\n");
System.out.println("Enter the new file name:");
String str2=in.readLine();
String str1,ss;
FileWriter f=new FileWriter(str2);
char buffer[];
while(true)
{
    str1=din.readLine();
    if(str1.equals("-1"))
    break;
    System.out.println(str1);
    buffer=new char[str1.length()];
    str1.getChars(0,str1.length(),buffer,0);
    f.write(buffer);
}
f.close();
clsct.close();
}
catch (Exception e)
{
System.out.println(e);
}
}
}

```

OUTPUT SERVER



```
Command Prompt - java Serverfile
Microsoft Windows [Version 10.0.19042.1165]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP>d:

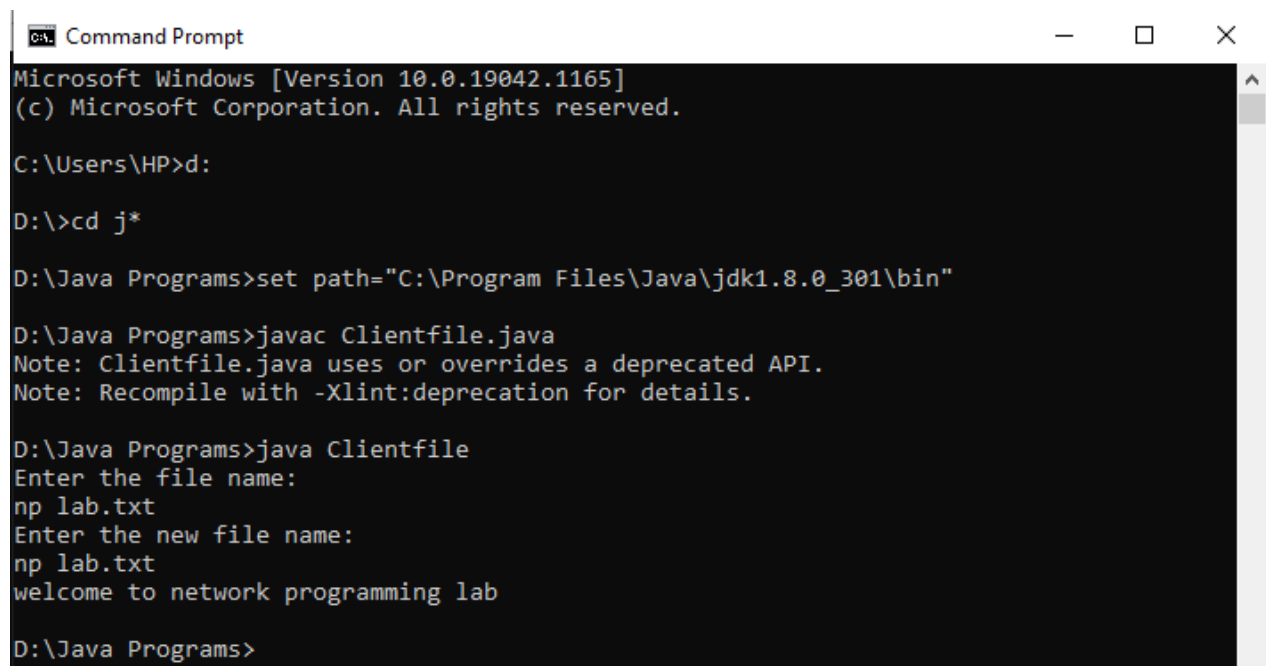
D:\>cd j*

D:\Java Programs>set path="C:\Program Files\Java\jdk1.8.0_301\bin"

D:\Java Programs>javac Serverfile.java
Note: Serverfile.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

D:\Java Programs>java Serverfile
welcome to network programming lab
```

CLIENT



```
Command Prompt
Microsoft Windows [Version 10.0.19042.1165]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP>d:

D:\>cd j*

D:\Java Programs>set path="C:\Program Files\Java\jdk1.8.0_301\bin"

D:\Java Programs>javac Clientfile.java
Note: Clientfile.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

D:\Java Programs>java Clientfile
Enter the file name:
np lab.txt
Enter the new file name:
np lab.txt
welcome to network programming lab

D:\Java Programs>
```

RESULT :

Thus the java program for implementation of DNS, SNMP and File Transfer application using TCP and UDP Sockets was executed successfully

EX.NO:7	Create a socket for HTTP for web page upload and download
DATE:	

AIM

To write a java program to create a socket for HTTP web page and download

ALGORITHM

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 request for image to be transfer
5. send the image to be print on the client
6. Close the socket.
7. End the program.

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 a connection ,request for image to be transfer .
4. Transfer the image and print it on client screen.
5. Close the socket.
6. End the program.

PROGRAM

//SERVER

```
import java.net.*;
import java.io.*;
import java.awt.image.*;
import javax.imageio.*;
import javax.swing.*;

class httpserver
{
public static void main(String args[]) throws Exception
{
ServerSocket server=null;
Socket socket;
server=new ServerSocket(4000);
System.out.println("Server Waiting for image");
socket=server.accept();
System.out.println("Client connected.");
InputStream in = socket.getInputStream();
DataInputStream dis = new DataInputStream(in);
int len = dis.readInt();
System.out.println("Image Size: " + len/1024 + "KB");
byte[] data = new byte[len];
dis.readFully(data);
dis.close();
in.close();
InputStream ian = new ByteArrayInputStream(data);
BufferedImage bImage = ImageIO.read(ian);
JFrame f = new JFrame("Server");
ImageIcon icon = new ImageIcon(bImage);
JLabel l = new JLabel();
l.setIcon(icon);
f.add(l);
f.pack();
f.setVisible(true);
}
}
```

```

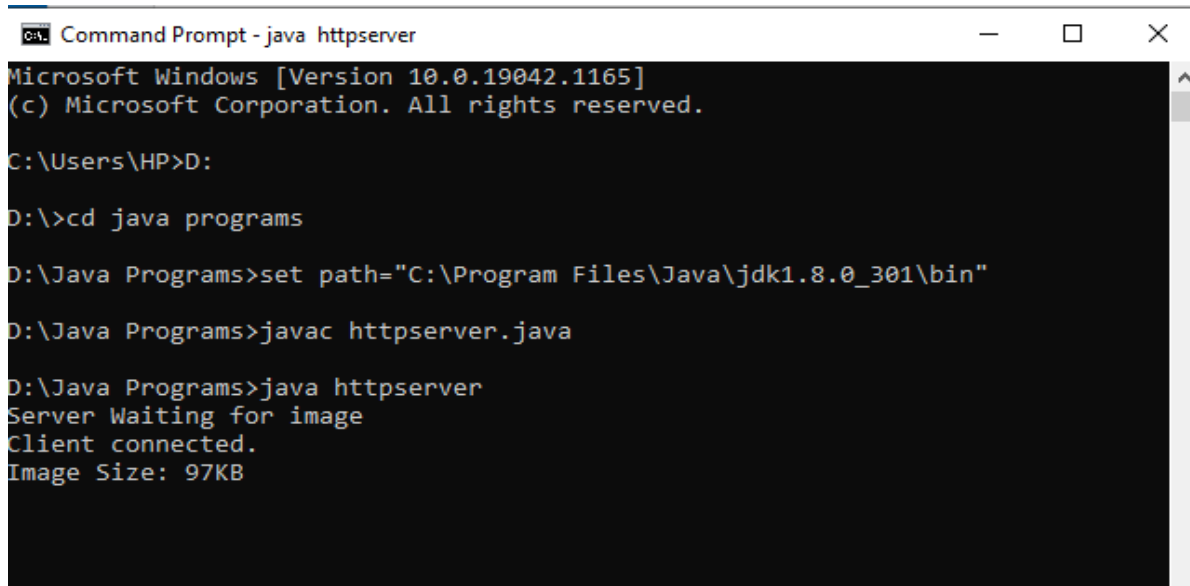
// CLIENT
import javax.swing.*;
import java.net.*;
import java.awt.image.*;
import javax.imageio.*;
import java.io.*;
import java.awt.image.BufferedImage;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;

public class httpclient
{
    public static void main(String args[]) throws Exception
    {
        Socket soc;
        BufferedImage img = null;
        soc=new Socket("localhost",4000);
        System.out.println("Client is running. ");
        try
        {
            System.out.println("Reading image from disk. ");
            img = ImageIO.read(new File("digital_image_processing.jpeg"));
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            ImageIO.write(img, "jpg", baos);
            baos.flush();
            byte[] bytes = baos.toByteArray();
            baos.close();
            System.out.println("Sending image to server. ");
            OutputStream out = soc.getOutputStream();
            DataOutputStream dos = new DataOutputStream(out);
            dos.writeInt(bytes.length);
            dos.write(bytes, 0, bytes.length);
            System.out.println("Image sent to server. ");
        }
    }
}

```

```
dos.close();
out.close();
}
catch (Exception e)
{
System.out.println("Exception: " + e.getMessage());
soc.close();
}
soc.close();
}
}
```

OUTPUT **SERVER**



```
C:\> Command Prompt - java httpserver
Microsoft Windows [Version 10.0.19042.1165]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP>D:

D:\>cd java programs

D:\Java Programs>set path="C:\Program Files\Java\jdk1.8.0_301\bin"

D:\Java Programs>javac httpserver.java

D:\Java Programs>java httpserver
Server Waiting for image
Client connected.
Image Size: 97KB
```

CLIENT

```
Command Prompt - java httpserver
Microsoft Windows [Version 10.0.19042.1165]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP>D:

D:\>cd java programs

D:\Java Programs>set path="C:\Program Files\Java\jdk1.8.0_301\bin"

D:\Java Programs>javac httpserver.java

D:\Java Programs>java httpserver
Server Waiting for image
Client connected.
Image Size: 97KB
```



RESULT

Thus the Java program to create socket for HTTP webpage upload and download was executed successfully.

EX.NO:8	Develop a program to Display the client's address at the server end
DATE:	

AIM:

To write and implement a java program to display the client's address at the server end.

ALGORITHM:**CLIENT**

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 accept 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

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
7. The server communicate the client to send the end of the message.
8. Stop the program

PROGRAM

//SERVER

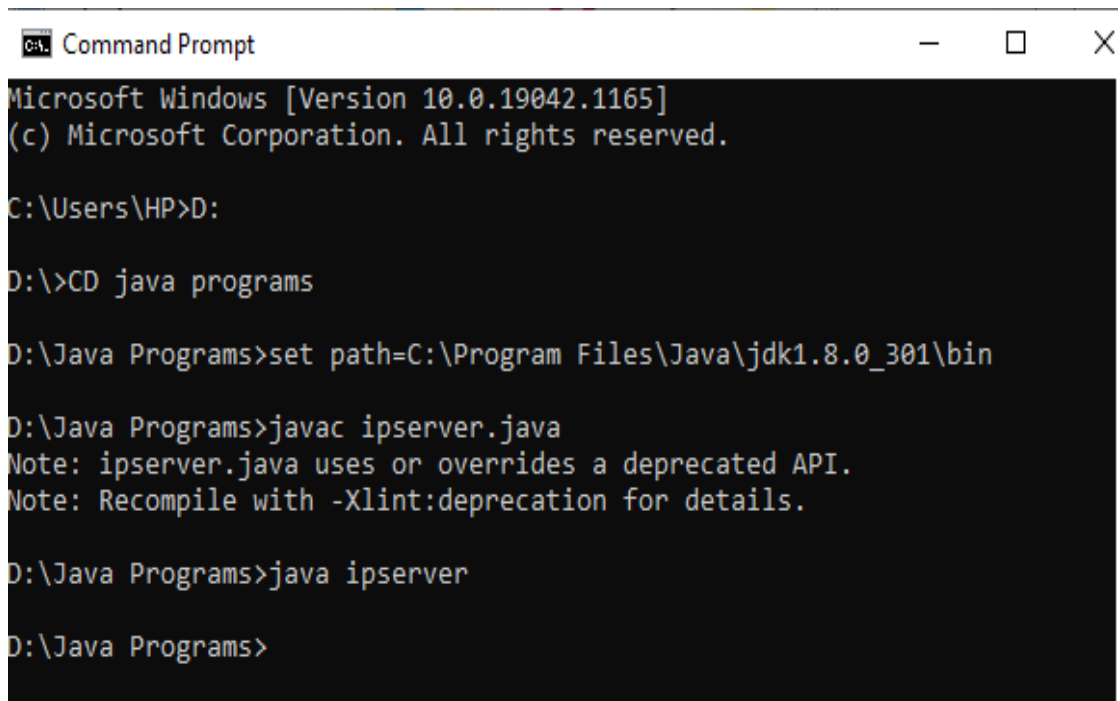
```
import java.io.*;
import java.net.*;
import java.util.*;
import java.text.*;
class ipserver
{
public static void main(String args[])throws Exception
{
ServerSocket ss=new ServerSocket(5555);
Socket s=ss.accept();
int c=0;
while(c<4)
{
DataInputStream dis=new DataInputStream(s.getInputStream());
PrintStream out=new PrintStream(s.getOutputStream());
String str=dis.readLine();
out.println("Reply from"+InetAddress.getLocalHost()+"Length"+str.length());
c++;
}
s.close();
}
```

//CLIENT

```
import java.io.*;
import java.net.*;
import java.util.Calendar;
class ipclient
{
public static void main(String args[])throws Exception
{
String str;
int c=0;
long t1,t2;
Socket s=new Socket("127.0.0.1",5555);
DataInputStream dis=new DataInputStream(s.getInputStream());
PrintStream out=new PrintStream(s.getOutputStream());
while(c<4)
{
```

```
t1=System.currentTimeMillis();
str="Welcome to network programming world";
out.println(str);
System.out.println(dis.readLine());
t2=System.currentTimeMillis();
System.out.println("TTL="+t2-t1+"ms");
c++;
}
s.close();
}
}
```

OUTPUT:
SERVER

A screenshot of a Windows Command Prompt window. The title bar reads "Command Prompt" with standard minimize, maximize, and close buttons. The window content shows the following text:

```
Microsoft Windows [Version 10.0.19042.1165]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP>D:

D:\>CD java programs

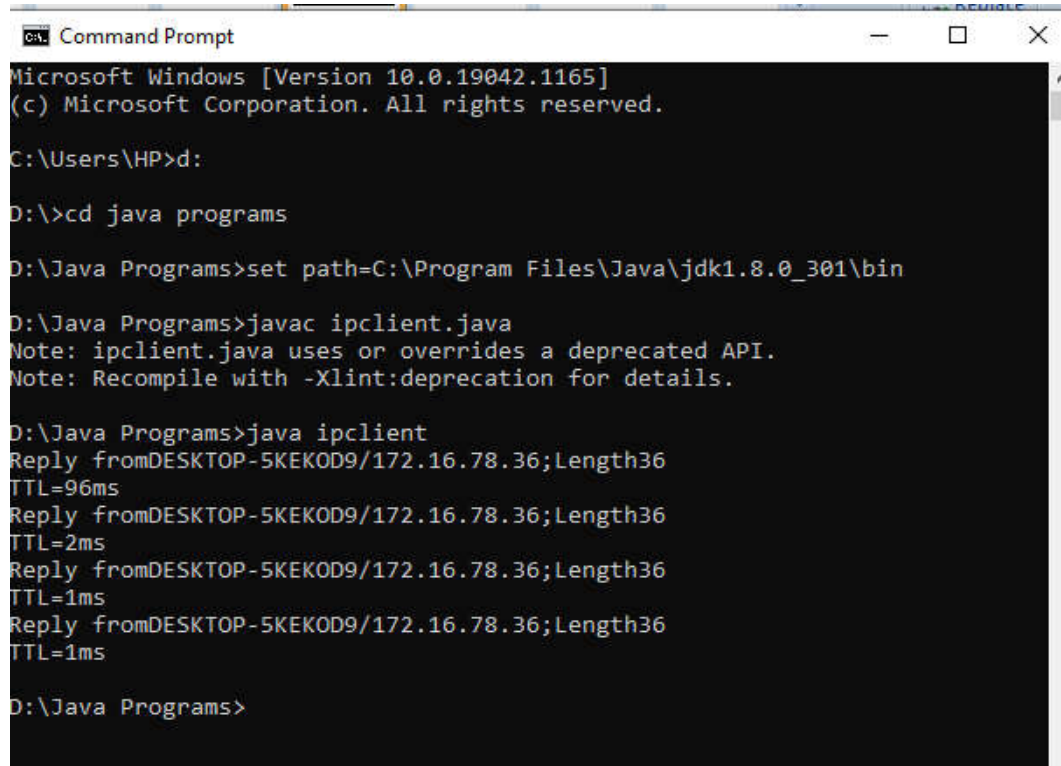
D:\Java Programs>set path=C:\Program Files\Java\jdk1.8.0_301\bin

D:\Java Programs>javac ipserver.java
Note: ipserver.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

D:\Java Programs>java ipserver

D:\Java Programs>
```

CLIENT



```
Command Prompt
Microsoft Windows [Version 10.0.19042.1165]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP>d:

D:\>cd java programs

D:\Java Programs>set path=C:\Program Files\Java\jdk1.8.0_301\bin

D:\Java Programs>javac ipclient.java
Note: ipclient.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

D:\Java Programs>java ipclient
Reply fromDESKTOP-5KEKOD9/172.16.78.36;Length36
TTL=96ms
Reply fromDESKTOP-5KEKOD9/172.16.78.36;Length36
TTL=2ms
Reply fromDESKTOP-5KEKOD9/172.16.78.36;Length36
TTL=1ms
Reply fromDESKTOP-5KEKOD9/172.16.78.36;Length36
TTL=1ms

D:\Java Programs>
```

RESULT

Thus the Java program to create socket for HTTP webpage upload and download was executed successfully.

EX.NO:9	Study of Network simulator (NS) and Simulation of Congestion Control Algorithms using NS
DATE:	

AIM:

To Study of Network Simulator(NS).and Simulation of Congestion Control Algorithms using NS.

NS OVERVIEW

- Ns programming:AQuickstart
- Case study I: A simple Wireless Network
- Boson NetSim

NS FUNCTIONALITIES

- Routing,
- Transportation,
- Traffic sources,
- Queuingdisciplines,
- QoS

WIRELESS

Ad hoc routing, mobile IP, sensor-MACTracing, visualization and various utilitieNS(NetworkSimulators)

Most of the commercial simulators are GUI driven, while some network simulators areCLI driven. The network model / configuration describes the state of the network (nodes,routers,switches, links) and the events (data transmissions, packet error etc.). An important output ofsimulations are the trace files. Trace files log every packet, every event that occurred in thesimulation and are used for analysis. Network simulators can also provide other tools to facilitatevisualanalysis of trendsand potentialtrouble spots.

Most network simulators use discrete event simulation,in which a list of pending"events" is stored, and those events are processed in order, with some events triggering future events—such as the event of the arrival of a packet at one node triggering the event of the arrival of that packet at a downstream node.

Simulation of networks is a very complex task. For example, if congestion is high, then estimation of the average occupancy is challenging because of high variance. To estimate the likelihood of a buffer overflow in a network, the time required for an accurate answer can be extremely large. Specialized techniques such as "control variates" and "importance sampling" have been developed to speed simulation.

EXAMPLES OF NETWORK SIMULATORS

There are many both free/open-source and proprietary network simulators. Examples of not able network simulation software are,ordered after how often they are mentioned in research papers:

1. NS(opensource)
2. OPNET(proprietarysoftware)
3. NetSim(proprietarysoftware)

USES OF NETWORK SIMULATORS

Network simulators serve a variety of needs. Compared to the cost and time involved in settingup an entire test bed containing multiple networked computers, routers and data links, network simulators are relatively fast and inexpensive. They allow engineers, researchers to test scenarios that might be particularly difficult or expensive to emulate using real hardware - for instance,simulating a scenario with several nodes or experimenting with a new protocol in the network.Network simulators are particularly useful in allowing researchers to test new networking protocols or changes to existing protocols in a controlled and reproducible environment.A typical network simulator encompasses a wide range of networking technologies and can help the users to build complex networks from basic building blocks such as a variety of nodes and links. With the help of simulators, one can design hierarchical networks using various types of nodes like computers, hubs, bridges, routers, switches, links, mobile units etc.

Various types of Wide Area Network (WAN) technologies like TCP, ATM, IP etc. and LocalArea Network (LAN) technologies like Ethernet, token rings etc., can all be simulated with atypical simulator and the user can test, analyze various standard results apart from devising somenovel protocol or strategy for routing etc. Network simulators are also widely used to simulate battle field networks in Network-centric warfare

There are a wide variety of network simulators, ranging from the very simple to the very complex. Minimally, a network simulator must enable a user to represent a network topology, specifying the nodes on the network, the links between those nodes and the traffic between the nodes. More complicated systems may allow the user to specify everything about the protocols used to handle traffic in a network. Graphical applications allow users to easily visualize the workings of their simulated environment. Text-based applications may provide a less intuitive interface, but may permit more advanced forms of customization.

PACKET LOSS

It occurs when one or more packets of data travelling across a computer network fail to reach their destination. Packet loss is distinguished as one of the three main error types encountered in digital communications;the other two being bit error and spurious packets caused due to noise.

Packets can be lost in a network because they may be dropped when a queue in the network node overflows. The amount of packet loss during the steady state is another important property of a congestion control scheme. The larger the value of packet loss, the more difficult it is for transport layer protocols to maintain high bandwidths, the

sensitivity to loss of individual packets, as well as to frequency and patterns of loss among longer packet sequences is strongly dependent on the application itself.

THROUGHPUT

This is the main performance measure characteristic, and most widely used. In communication networks, such as Ethernet or packet radio, throughput or network throughput is the average rate of successful message delivery over a communication channel. The throughput is usually measured in bits per second (bit/s or bps), and sometimes in data packets per second or data packets per time slot. This measure shows how soon the receiver is able to get a certain amount of data sent by the sender. It is determined as the ratio of the total data received to the end-to-end delay. Throughput is an important factor which directly impacts the network performance.

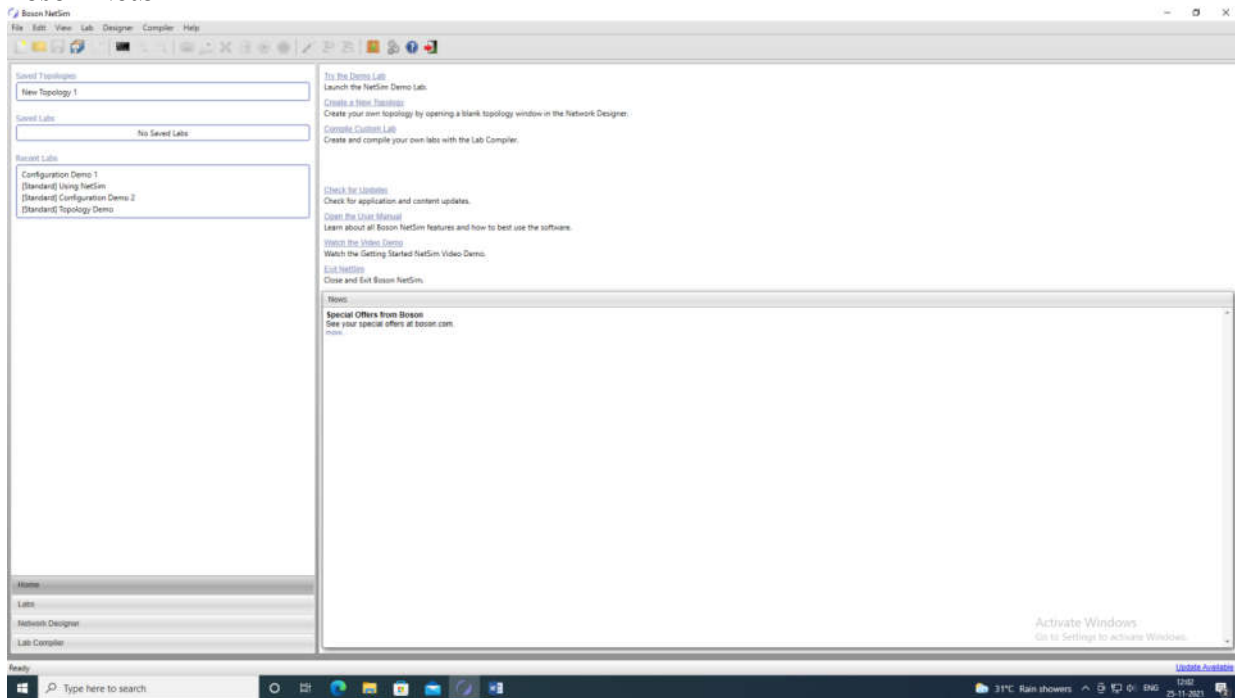
DELAY

Delay is the time elapsed while a packet travels from one point e.g., source premise or networking destination premise or network degrees. The larger the value of delay, the more difficult it is for transport layer protocols to maintain high bandwidths. We will calculate end-to-end delay

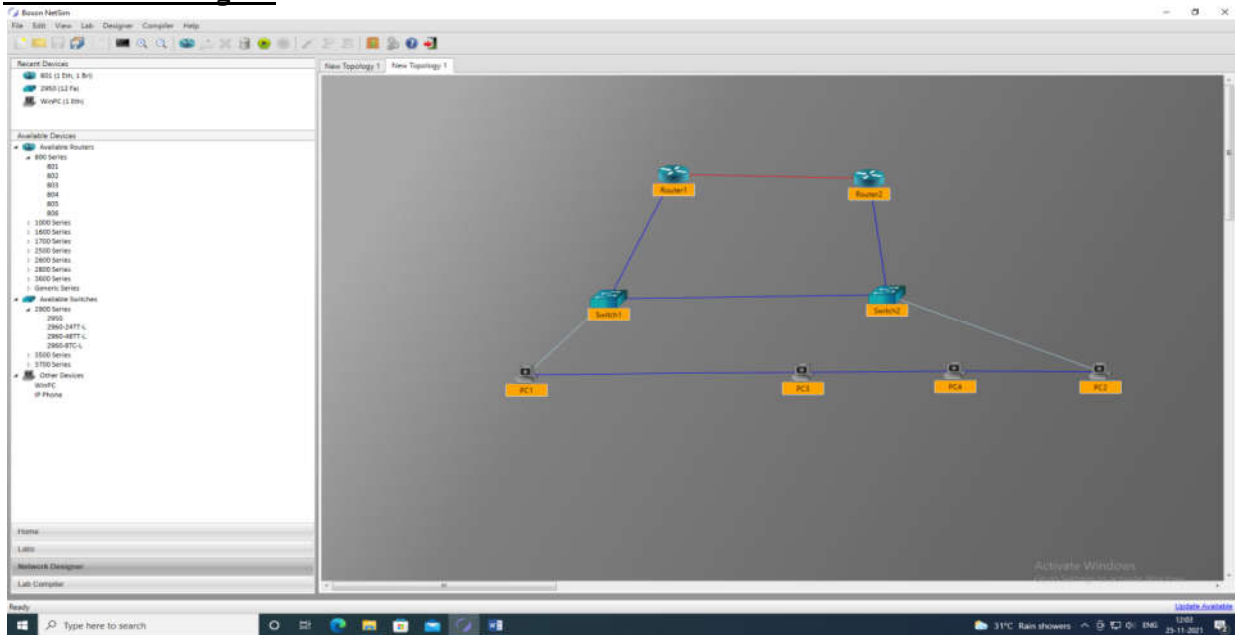
QUEUE LENGTH

A queuing system in networks can be described as packets arriving for service, waiting for service if it is not immediate, and if having waited for service, leaving the system after being served. Thus queue length is very important characteristic to determine that how well the active queue management of the congestion control algorithm has been working.

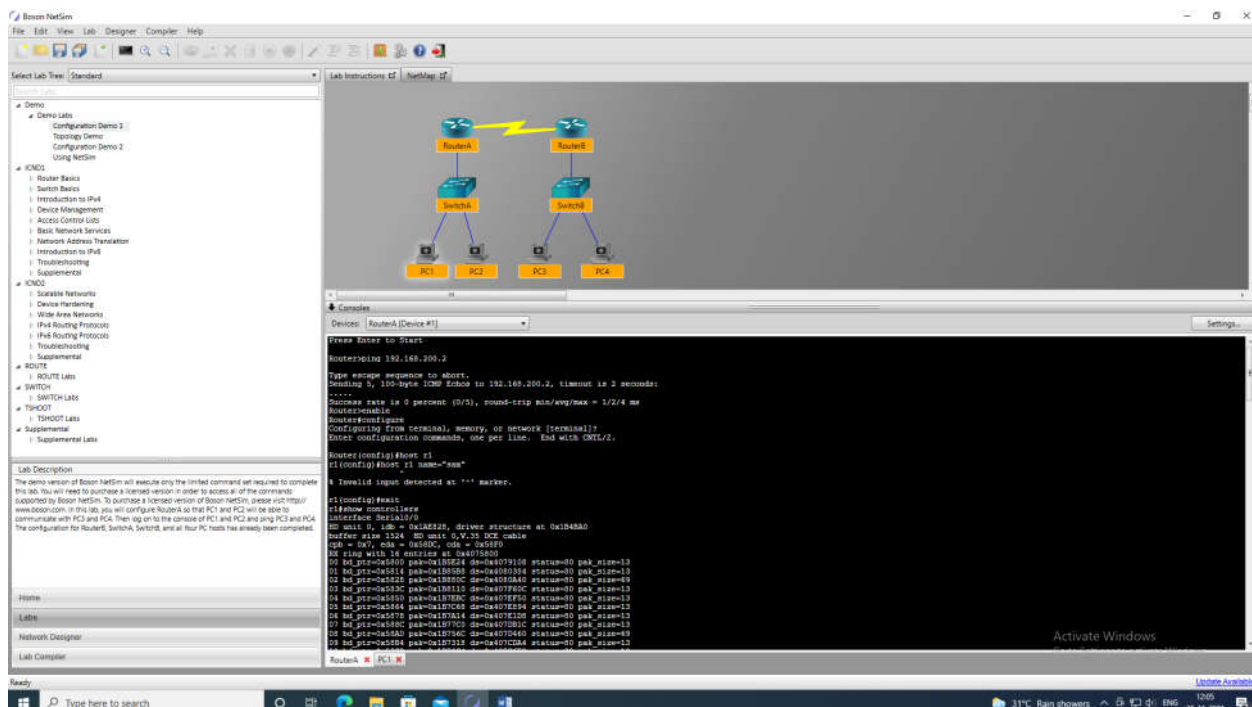
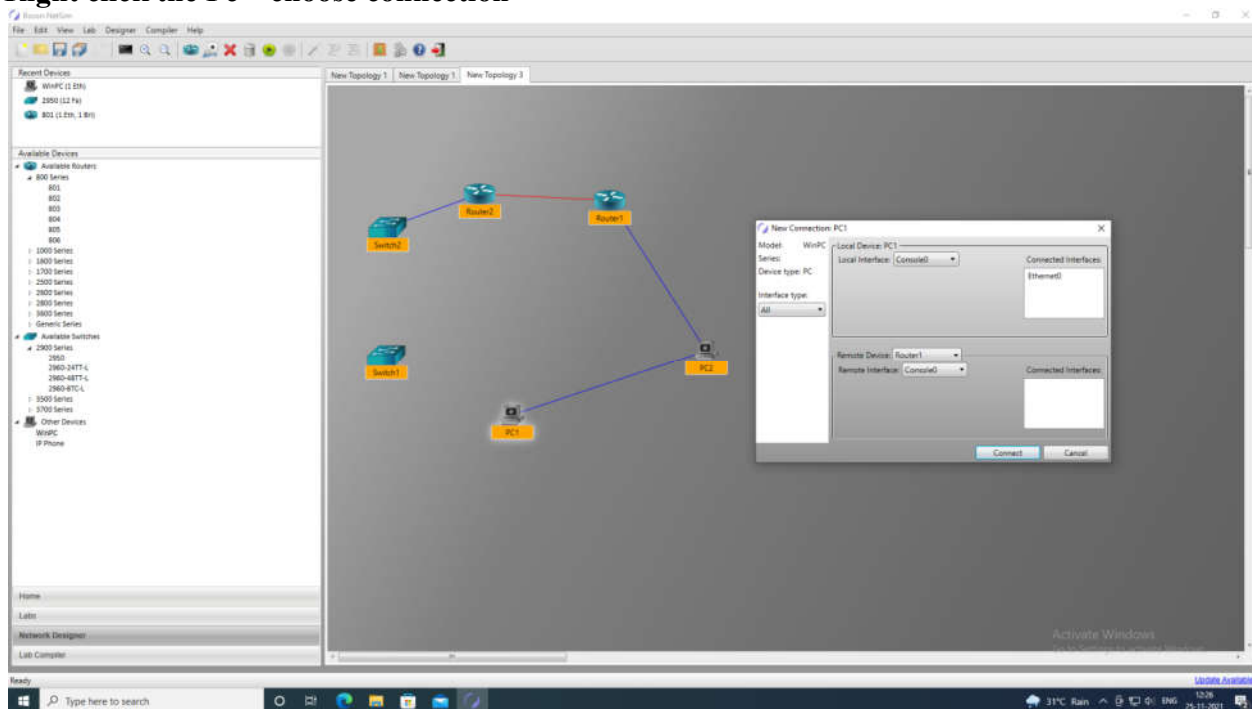
Boson NetSim



1. Network Designer



2. Connections:



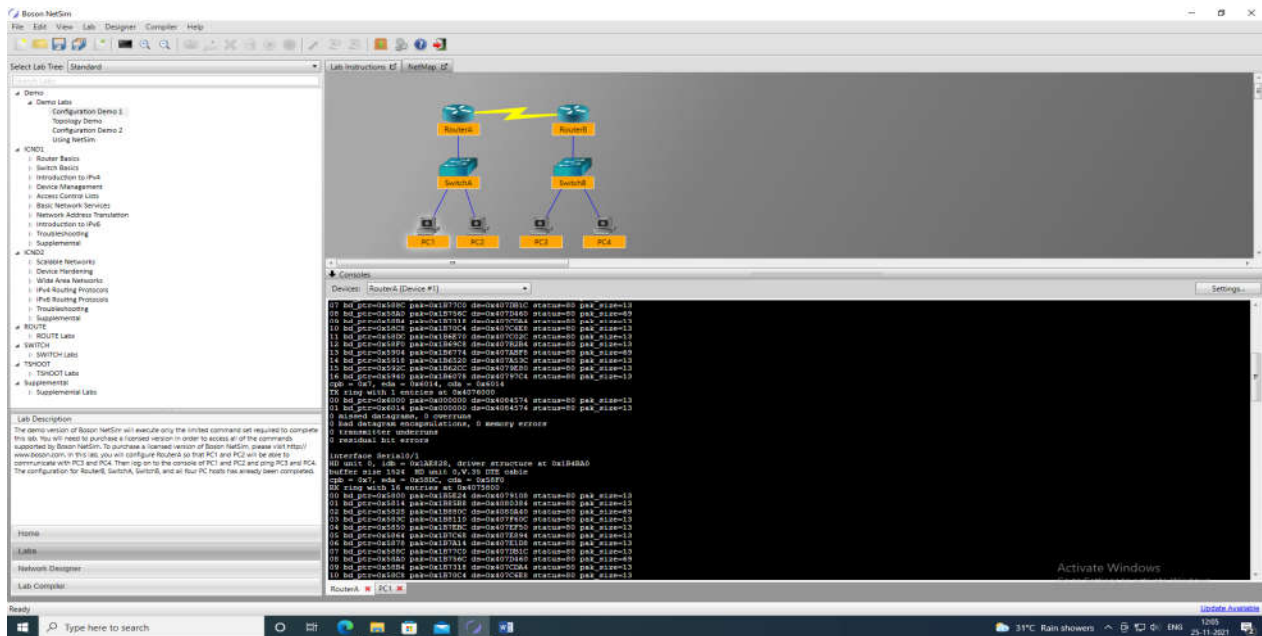
3. Console Chooserouter:

Router> ping 192.168.200.2

>host r1

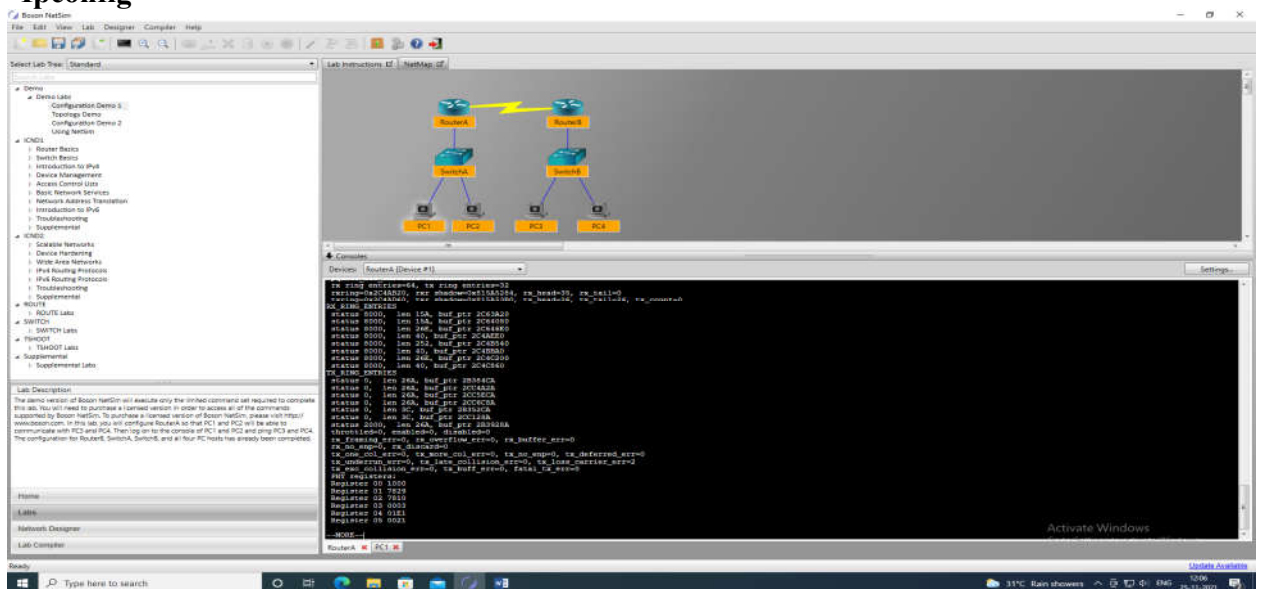
>r1 config

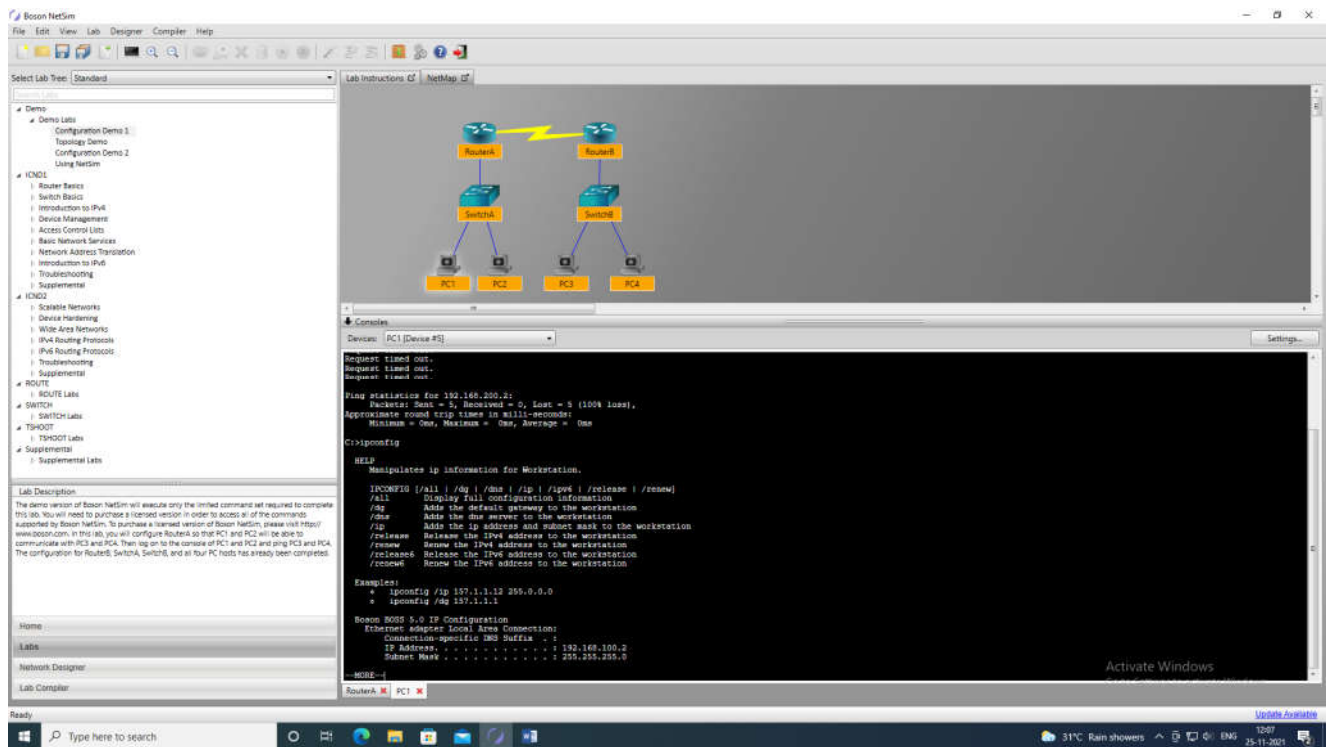
>exit



4.Pc command

>Ipconfig





RESULT

Thus the study of Network Simulator(NS2) was studied.

EX.NO:10	Perform a case study about the different routing algorithms to select the network path with its optimum and economical during data transfer.
DATE:	

i. Link State routing

AIM:

To study the link state routing

Link State routing

Routing is the process of selecting best paths in a network. In the past, the term routing was also used to mean forwarding network traffic among networks. However this latter function is much better described as simply forwarding. Routing is performed for many kinds of networks, including the telephone network (circuit switching), electronic data networks (such as the Internet), and transportation networks. This article is concerned primarily with routing in electronic data networks using packet switching technology.

In packet switching networks, routing directs packet forwarding (the transit of logically addressed network packets from their source toward the ultimate destination) through intermediate nodes. Intermediate nodes are typically network hardware devices such as routers, bridges, gateways, firewalls, or switches. General-purpose computers can also forward packets and perform routing, though they are not specialized hardware and may suffer from limited performance. The routing process usually directs forwarding on the basis of routing tables which maintain a record of the routes to various network destinations. Thus, constructing routing tables, which are held in the router's memory, is very important for efficient routing. Most routing algorithms use only one network path at a time. Multipath routing techniques enable the use of multiple alternative paths.

In case of overlapping/equal routes, the following elements are considered in order to decide which routes get installed into the routing table (sorted by priority):

1. *Prefix-Length*: where longer subnet masks are preferred (independent of whether it is within a routing protocol or over different routing protocol)
2. *Metric*: where a lower metric/cost is preferred (only valid within one and the same routing protocol)
3. *Administrative distance*: where a lower distance is preferred (only valid between different routing protocols)

Routing, in a more narrow sense of the term, is often contrasted with bridging in its assumption that network addresses are structured and that similar addresses imply proximity within the network. Structured addresses allow a single routing table entry to represent the route to a group of devices. In large networks, structured addressing (routing, in the narrow sense) outperforms unstructured addressing (bridging). Routing has become the dominant form of addressing on the Internet. Bridging is still widely used within localized environments.

ii. Flooding

Flooding is a simple routing algorithm in which every incoming packet is sent through every outgoing link except the one it arrived on. Flooding is used in bridging and in systems such as Usenet and peer-to-peer file sharing and as part of some routing protocols, including OSPF, DVMRP, and those used in ad-hoc wireless networks. There are generally two types of flooding available, Uncontrolled Flooding and Controlled Flooding. Uncontrolled Flooding is the fatal law of flooding. All nodes have neighbours and route packets indefinitely. More than two neighbours creates a broadcast storm.

Controlled Flooding has its own two algorithms to make it reliable, SNCF (Sequence Number Controlled Flooding) and RPF (Reverse Path Flooding). In SNCF, the node attaches its own address and sequence number to the packet, since every node has a memory of addresses and sequence numbers. If it receives a packet in memory, it drops it immediately while in RPF, the node will only send the packet forward. If it is received from the next node, it sends it back to the sender.

Algorithm

There are several variants of flooding algorithm. Most work roughly as follows:

1. Each node acts as both a transmitter and a receiver.
2. Each node tries to forward every message to everyone of its neighbours except the source node.

This results in every message eventually being delivered to all reachable parts of the network.

Algorithms may need to be more complex than this, since, in some case, precautions have to be taken to avoid wasted duplicate deliveries and infinite loops, and to allow messages to eventually expire from the system. A variant of flooding called *selective flooding* partially addresses these issues by only sending packets to routers in the same direction. In selective flooding the routers don't send every incoming packet on every line but only on those lines which are going approximately in the right direction.

Advantages

- A packet can be delivered, it will (probably multiple times).
- Since flooding naturally utilizes every path through the network, it will also use the shortest path.
- This algorithm is very simple to implement.

Dis advantages

- Flooding can be costly in terms of wasted bandwidth. While a message may only have one destination it has to be sent to every host. In the case of a ping flood or a denial of service attack, it can be harmful to the reliability of a computer network.
- Messages can become duplicate in the Network further increasing the load on the network's bandwidth as well as requiring an increase in processing complexity to disregard duplicate messages.
- Duplicate packets may circulate forever, unless certain precautions are taken:
- Use a hop count or a time to live count and include it with each packet. This value should take into account the number of nodes that a packet may have to pass through on the way to its destination.
- Have each node keep track of every packet seen and only forward each packet once
- Enforce a network topology without loops

iii.Distance Vector

In computer communication theory relating to packet-switched networks, a **distance-vector routing protocol** is one of the two major classes of routing protocols, the other major class being the link-state protocol. Distance-vector routing protocols use the Bellman–Ford algorithm, Ford–Fulkerson algorithm, or DUAL FSM (in the case of Cisco Systems's protocols) to calculate paths.

A distance-vector routing protocol requires that a router informs its neighbors of topology changes periodically. Compared to link-state protocols, which require a router to inform all the nodes in a network of topology changes, distance-vector routing protocols have less computational complexity and message overhead.

The term *distance vector* refers to the fact that the protocol manipulates *vectors* (arrays) of distances to other nodes in the network. The vector distance algorithm was the original ARPANET routing algorithm and was also used in the internet under the name of RIP (Routing Information Protocol).

Examples of distance-vector routing protocols include RIPv1 and RIPv2 and IGRP.

Method

Routers using distance-vector protocol do not have knowledge of the entire path to a destination. Instead they use two methods:

1. Direction in which router or exit interface a packet should be forwarded.
2. Distance from its destination

Distance-vector protocols are based on calculating the direction and distance to any link in a network. "Direction" usually means then ext hop address and the exit interface. "Distance" is a measure of the cost to reach a certain node. The least cost route between any two nodes is the route with minimum distance. Each node maintains a vector (table) of minimum distance to every node. The cost of reaching a destination is calculated using various route metrics. RIP uses the hop count of the destination whereas IGRP takes into account other information such as node delay and available bandwidth.

Updates are performed periodically in a distance-vector protocol where all or part of a router's routing table is sent to all its neighbors that are configured to use the same distance-vector routing protocol. RIP supports cross-platform distance vector routing whereas IGRP is a Cisco Systems proprietary distance vector routing protocol. Once a router has this information it is able to amend its own routing table to reflect the changes and then inform its neighbors of the changes. This process has been described as routing by rumor because routers are relying on the information they receive from other routers and cannot determine if the information is actually valid and true. There are a number of features which can be used to help with instability and in accurate routing information.

EGP and BGP are not pure distance-vector routing protocols because a distance-vector protocol calculates routes based only on link costs whereas in BGP, for example, the local route preference value takes priority over the link cost.

Count-to-infinity problem

The Bellman–Ford algorithm does not prevent routing loops from happening and suffers from the **count-to-infinity problem**. The core of the count-to-infinity problem is that if A tells B that it has a path somewhere, there is no way for B to know if the path has B as a part of it. To see the problem clearly, imagine a subnet connected like A–B–C–D–E–F, and let the metric between the routers be "number of jumps". Now suppose that A is taken offline. In the vector-update-process B notices that the route to A, which was distance 1, is down – B does not receive the vector update from A. The problem is, B also gets an update from C, and C is still not aware of the fact that A is down – so it tells B that A is only two jumps from C (C to B to A), which is false. This slowly propagates through the network until it reaches infinity (in which case the algorithm corrects itself, due to the relaxation property of Bellman–Ford).

RESULT

Thus the Perform a case study about the different routing algorithms to select the network path with its optimum and economical during data transfer was completed.