INDEX

EXPERIMENT 1:

IMPLEMENTATION OF STOP AND WAIT PROTOCOL AND SLIDING WINDOW PROTOCOL.

(I) STOP AND WAIT

PROTOCOL Objective:
To write a java program to perform Stop and Wait protocol.

Algorithm:

Step1: Start the program.
Step2: Create the socket by specifying the address and establishes the connection
Step3: Send and receive information.
Step4: The sender sends one frame, stops until it receives confirmation from the receiver and then sends the next frame.
Step5: If your frames reach the server it will send ACK signal to client otherwise it will send NACK signal to client.
Step6: Stop the program

Sample Code:

**Sender.jav
a**

```
import
java.io.*;
import
java.net.*;
BufferedReader br=new  BufferedReader(newInputStreamReader(System.in));
System.out.println("Waiting for Connection.  ");
sender=new Socket("localhost",2004);

 sequence=0;

out=new ObjectOutputStream(sender.getOutputStream());
out.flush();
in=new ObjectInputStream(sender.getInputStream());
str=(String)in.readObject();
System.out.println("reciver> "+str);
System.out.println("Enter the data to send ... ");
packet=br.readLine();

n=packet.lengt
```

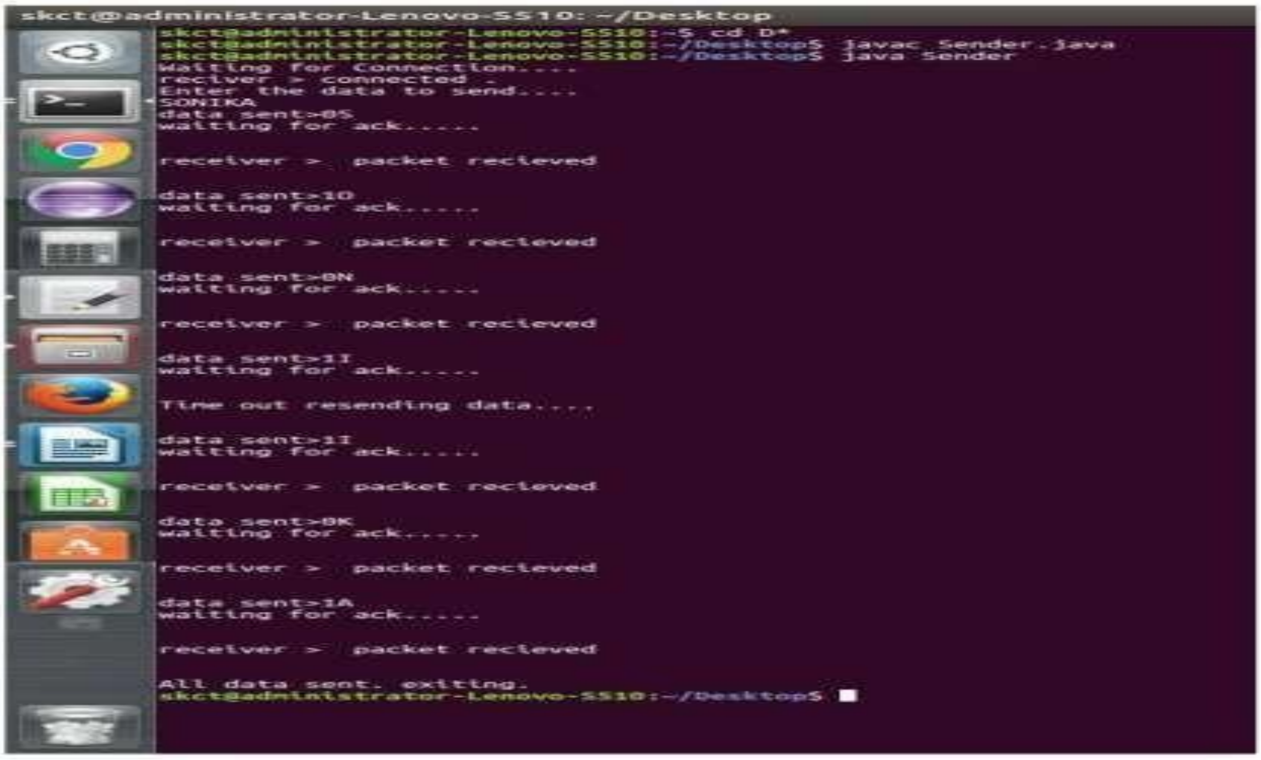
```
h (); do{try{
if(i<n){
msg=String.valueOf(sequence);
msg=msg.concat(packet.substring(i,i
+1));}
else if(i==n){
msg="end";
out.writeObject(msg);
break;}
 out.writeObject(msg);
sequence=(sequence==0?1:0;
 out.flush();
```

Reciever.java

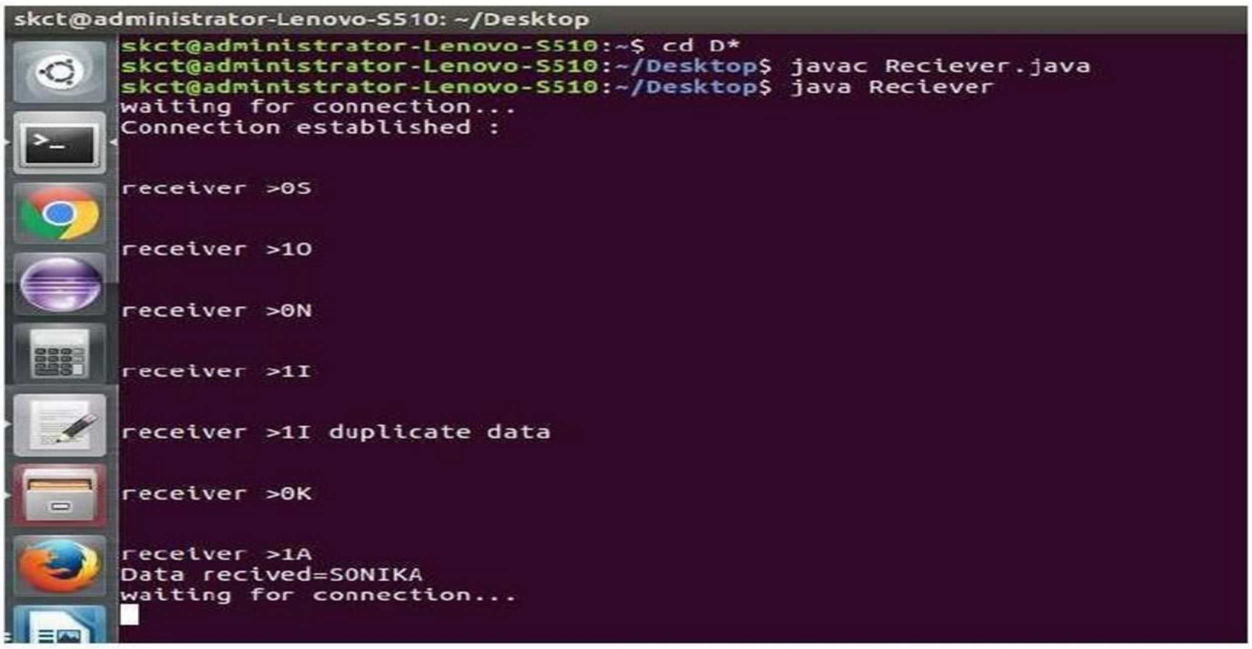
```
ServerSocketreciever;
 Socket connection=null;
ObjectOutputStream out;
ObjectInputStream in;
public void run(){

try{
BufferedReader br=newBufferedReader(new InputStreamReader(System.in));
 reciever = new ServerSocket(2004,10);
System.out.println("waiting for connection. . ");

connection=reciever.accept();
sequence=0;
System.out.println("Connection established :");

out=new ObjectOutputStream(connection.getOutputStream());
out.flush(); in=new
ObjectInputStream(connection.getInputStream());

out.writeObject("connected .");
do{ try{
packet=(String)in.read
Object();
if(Integer.valueOf(packet.substring(0,1))==sequence){
data+=packet.substring(1);
 sequence=(sequence==0)?1:0;
System.out.println("\n\nreceiver>"+packet); }
else{System.out.println("\n\nreceiver>"+packet +" duplicate data"); }
if(i<3){
out.writeObject(String.valueOf(sequence));
i++; }
else
```

```
out.writeObject(String.valueOf((sequence+1)%2));
```

Sample Output:



```
skct@administrator-Lenovo-S510: ~/Desktop
skct@administrator-Lenovo-S510:~$ cd D*
skct@administrator-Lenovo-S510:~/Desktop$ javac Sender.java
skct@administrator-Lenovo-S510:~/Desktop$ java Sender
Waiting for Connection....
reciver > connected .
Enter the data to send....
SONIKA
data sent>0S
waiting for ack.....

receiver >  packet recieved

data sent>1O
waiting for ack.....

receiver >  packet recieved

data sent>0N
waiting for ack.....

receiver >  packet recieved

data sent>1I
waiting for ack.....

Time out resending data....

data sent>1I
waiting for ack.....

receiver >  packet recieved

data sent>0K
waiting for ack.....

receiver >  packet recieved

data sent>1A
waiting for ack.....

receiver >  packet recieved

All data sent. exiting.
skct@administrator-Lenovo-S510:~/Desktop$ 
```



```
skct@administrator-Lenovo-S510: ~/Desktop
skct@administrator-Lenovo-S510:~$ cd D*
skct@administrator-Lenovo-S510:~/Desktop$ javac Reciever.java
skct@administrator-Lenovo-S510:~/Desktop$ java Reciever
waiting for connection...
Connection established :


receiver >0S


receiver >1O


receiver >0N


receiver >1I


receiver >1I duplicate data


receiver >0K


receiver >1A
Data recived=SONIKA
waiting for connection...
```

(II) SLIDING WINDOW PROTOCOL

Objective:
To write a java program to perform sliding window.

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 client. 6. Stop the program

Program :

```
import java.net.*;

import java.io.*;
import java.rmi.*;
public class slidsender {
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.read
Line()); 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("y
es")); s.close();

}
}
```

RECEIVER PROGRAM

```
import java.net.*;
import java.io.*;
class slidreceiver {
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.readLin
e()); if(nf<=rws-1)
 {
for(i=1;i<=nf;i++)
{
```

```
rptr=++rptr%8;
rbuf[rptr]=in.readL
ine();
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
{
bre
ak;
}
ch=in.readLine();
}
while(ch.equals("yes"));
}
}
```

OUTPUT:

//SENDER OUTPUT

```
Enter the no. of frames : 4 Enter 4 Messages to be send
 Hiii how r u  i
am fine how is
everyone
Acknowledgment received for 4 frames
Do you wants to send some more frames : no
```

//RECEIVER OUTPUT

```
The received Frame
0 is : hiii The
received Frame 1 is
: how r u
The received Frame 2 is : i am fine
The received Frame 3 is : how is everyone
```

Signature:__________________

EXPERIMENT 2:
STUDY OF SOCKET PROGRAMMING AND CLIENT – SERVER MODEL

Objective:
To implement socket programming date and time display from client to server using TCP and UDP Sockets.

Algorithm:
Server:
Step1: Create a server socket and bind it to port.
Step 2: Listen for new connection and when a connection arrives, accept it. **Step 3:**Send server's date and time to the client. **Step 4:**Read client's IP address sent by the client. **Step 5:**Display the client details.
Step 6: Repeat steps 2-5 until the server is terminated.
Step 7: Close the server socket.

Client:
Step1: Create a client socket and connect it to the server's port number.
Step2: Retrieve its own IP address using built-in function.
Step3: Send its address to the server.
Step4: Display the date & time sent by the server. **Step5:**Close the client socket
System and Software tools required: Package Required : Java Compiler Operating System : UBUNTU
Minimum Requirement : Pentium III or Pentium IV with 2GB RAM 40 GB hard disk

TCP Program:
dateserver.java
```
import java.net.*;
import java.io.*;
 importjava.util.*;
ss=new ServerSocket(8020);
s=ss.accept();
ps=new PrintStream(s.getOutputStream());
Date d=new Date();
ps.println(d);
dis=new DataInputStream(s.getInputStream());
 inet=dis.readLine();
System.out.println("THE CLIENT SYSTEM ADDRESS IS :"+inet);
```

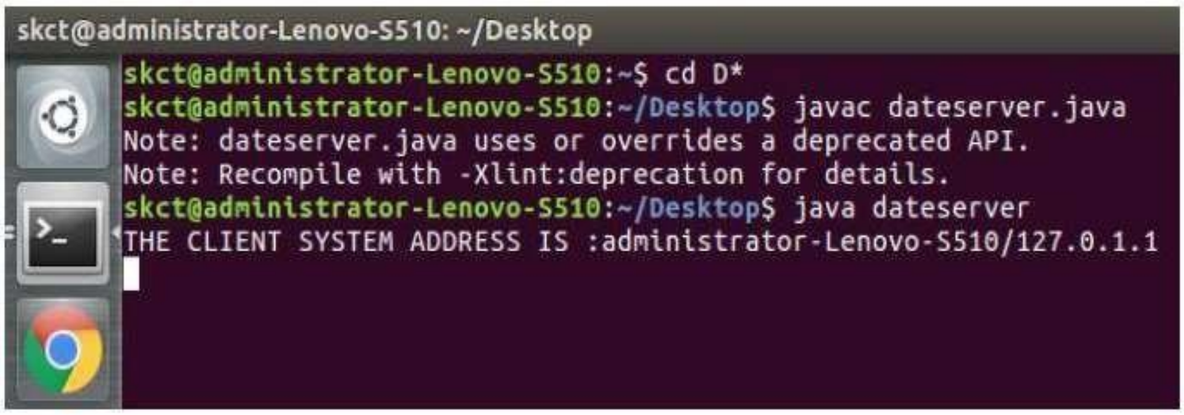
ps.close();}}

dateclient.java

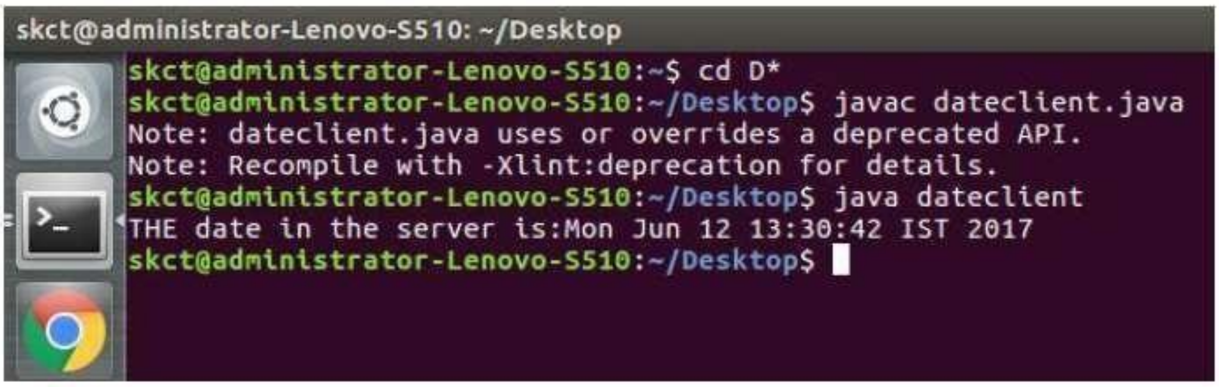
Socket soc;
DataInputStrea
m dis; String
sdate;
PrintStreamps;
InetAddressia=InetAddress.getLocalHost();
 soc=new Socket(ia,8020);
dis=new DataInputStream(soc.getInputStream());
 sdate=dis.readLine();

System.out.println("THE date in the server
is:"+sdate); ps=new
PrintStream(soc.getOutputStream());
ps.println(ia);}

Output:



```
skct@administrator-Lenovo-S510: ~/Desktop
skct@administrator-Lenovo-S510:~$ cd D*
skct@administrator-Lenovo-S510:~/Desktop$ javac dateserver.java
Note: dateserver.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
skct@administrator-Lenovo-S510:~/Desktop$ java dateserver
THE CLIENT SYSTEM ADDRESS IS :administrator-Lenovo-S510/127.0.1.1
```



```
skct@administrator-Lenovo-S510: ~/Desktop
skct@administrator-Lenovo-S510:~$ cd D*
skct@administrator-Lenovo-S510:~/Desktop$ javac dateclient.java
Note: dateclient.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
skct@administrator-Lenovo-S510:~/Desktop$ java dateclient
THE date in the server is:Mon Jun 12 13:30:42 IST 2017
skct@administrator-Lenovo-S510:~/Desktop$
```

Server.java

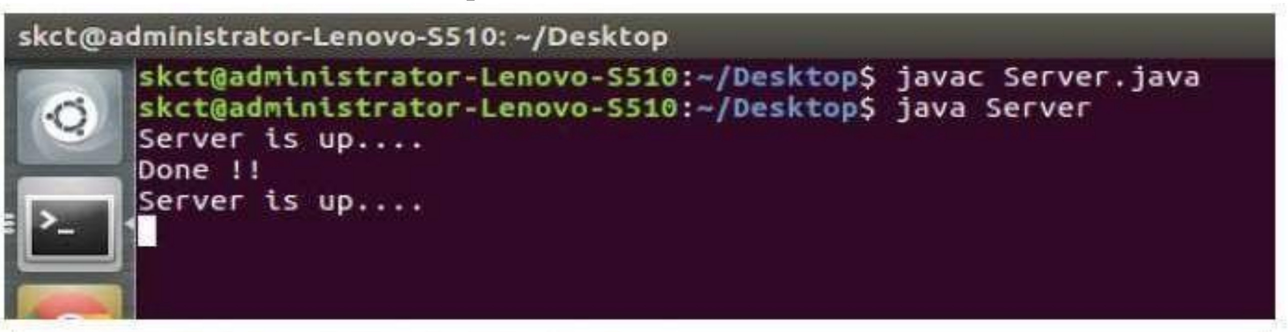
import java.net.*; import java.io.*;
importjava.util.*; DatagramPacketrp=new
DatagramPacket(rd,rd.length); ss.receive(rp);
InetAddress ip= rp.getAddress(); int

port=rp.getPo
rt(); Date
d=new
Date(); String
time= d + "";
sd=time.getB
ytes();
DatagramPacketsp=new
DatagramPacket(sd,sd.length,ip,port); ss.send(sp);

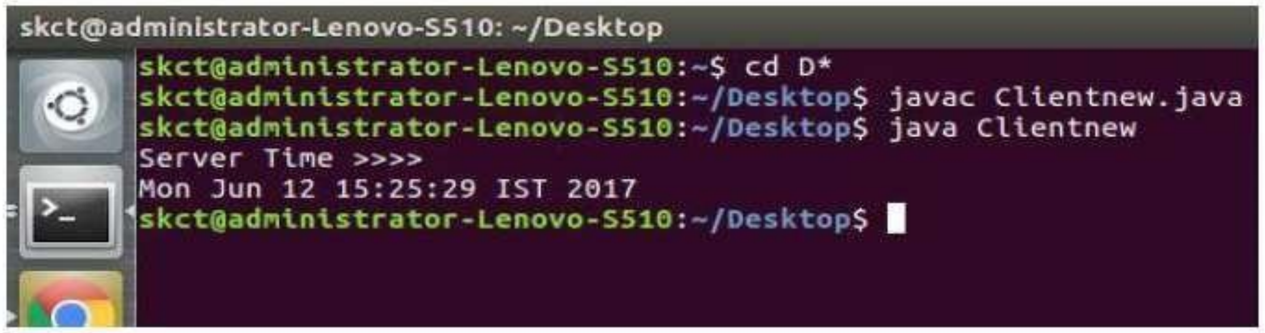
Clientnew.java
DatagramPacketsp=new DatagramPacket(sd,sd.length,ip,1234);
DatagramPacketrp=new DatagramPacket(rd,rd.length);

cs.send(sp); cs.receive(rp);
String time=new String(rp.getData());
 System.out.println(time);
cs.close(); } }

Output:



```
skct@administrator-Lenovo-S510: ~/Desktop
skct@administrator-Lenovo-S510:~/Desktop$ javac Server.java
skct@administrator-Lenovo-S510:~/Desktop$ java Server
Server is up....
Done !!
Server is up....
```



```
skct@administrator-Lenovo-S510: ~/Desktop
skct@administrator-Lenovo-S510:~$ cd D*
skct@administrator-Lenovo-S510:~/Desktop$ javac Clientnew.java
skct@administrator-Lenovo-S510:~/Desktop$ java Clientnew
Server Time >>>>
Mon Jun 12 15:25:29 IST 2017
skct@administrator-Lenovo-S510:~/Desktop$
```

Signature:________________

EXPERIMENT 3:
WRITE A CODE SIMULATING ARP /RARP PROTOCOLS.

OBJECTIVE:
To write a java program for simulating arp/rarp protocols

ALGORITHM:

server

1. Create a server socket and bind it to port.
2. Listen for new connection and when a connection arrives, accept it.
3. Send server"s date and time to the client.
4. Read client"s IP address sent by the client.
5. Display the client details.
6. Repeat steps 2-5 until the server is terminated.
7. Close all streams.
8. Close the server socket.
9. Stop.

Client

1. Create a client socket and connect it to the server"s port number.
2. Retrieve its own IP address using built-in function.

3. Send its address to the server.
4. Display the date & time sent by the server.
5. Close the input and output streams.
6. Close the client socket.
7. Stop.

(i) Program for Address Resolutuion Protocol (ARP) using

TCP Client:

```
import java.io.*;
 import java.net.*;
 import java.util.*;
 class Clientarp {
public static void main(String args[])
{ try
{
BufferedReader in=new BufferedReader(new InputStreamReader(System.in));
 Socket clsct=new Socket("127.0.0.1",139);
DataInputStream din=new DataInputStream(clsct.getInputStream());
DataOutputStream dout=new DataOutputStream(clsct.getOutputStream());
System.out.println("Enter the Logical address(IP):");
String str1=in.readLine();
dout.writeBytes(str1
+'\n');
String  str=din.readLine();
System.out.println("The Physical Address is: "+str);
 clsct.close();
}
catch (Exception e)
{
System.out.println(e);
}
}
}
```

Server
:

```
 import
java.io.*;
import
java.net.*;
import
java.util.*;
class
Serverarp {
public static void main(String args[])
{ try
{
ServerSocket obj=new ServerSocket(139);
Socket obj1=obj.accept();
while(true)
{
DataInputStream din=new DataInputStream(obj1.getInputStream());
DataOutputStream dout=new
DataOutputStream(obj1.getOutputStream());
String str=din.readLine();
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(str.equals(ip[i]))
{
dout.writeBytes(mac[i]
+'\n');
break;
} }
obj.close();
}
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

Output:

```
E:\networks>java
Serverarp
E:\networks>java
Clientarp
Enter the Logical
address(IP):
165.165.80.80
The Physical Address is: 6A:08:AA:C2
```

(ii) Program for Reverse Address Resolutuion Protocol (RARP) using

UDP
Client:

```
import
java.io.*;
import
java.net.*;
import
java.util.*;
class
Clientrarp1
2
{
public static void main(String args[])

{
t
r
y
{
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(MAC):");
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 Logical Address is(IP): "+s.trim());
 client.close();
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

Server:

```
Import java.io.*;
 import java.net.*;
import java.util.*;
class Servezrarp12
{
public static void main(String args[])
{ try
{
DatagramSocket server=new DatagramSocket(1309);
 while(true)
{
byte[] sendbyte=new byte[1024]; byte[]
receivebyte=new byte[1024];
DatagramPacketreceiver=newDatagramPacket(receivebyte,receivebyte.le
ngth);

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(mac[i]))
{
sendbyte=ip[i].getBytes();
DatagramPacketsender=newDatagramPacket(sendbyte,sendbyte.length,addr,
port);
 server.send(sender); break;
} } break; } }
catch(Exception e)
{
System.out.println(e);
}
}
}
```

Output:

```
I:\ex>java
Serverrarp12
I:\ex>java
Clientrarp12
Enter the Physical address
(MAC): 6A:08:AA:C2
The Logical Address is(IP): 165.165.80.80
```

Signature__________________________

EXPERIMENT 4:
CREATE A SOCKET FOR HTTP FOR WEB PAGE UPLOAD AND DOWNLOAD.

OBJECTIVE:

To write a java program for socket for HTTP for web page upload and download
.

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 client. 6. Stop the program

Program :

//CLIENT CLASS

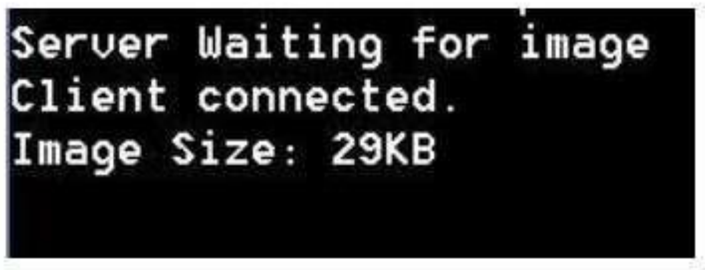
```
import javax.swing.*;
 import java.net.*;
import java.awt.image.*;
import javax.imageio.*;
import java.io.*;
import java.awt.image.BufferedImage;
import java.io.ByteArrayOutputStream;
 import java.io.File;
 import
java.io.IOException;
import
javax.imageio.ImageIO;
public class Client{
public static void main(String args[]) throws Exception{
Socket soc;
BufferedImage img = null;
soc=newSocket("localhost",4000);
System.out.println("Client is running. ");
  try {
System.out.println("Reading image from disk. ");
img=ImageIO.read(newFile("digital_image_processing.jpg"));
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();
}
}
```

//SERVER CLASS

```
import java.net.*;
 import java.io.*;
 import  java.awt.image.*;
 import javax.imageio.*;
 import javax.swing.*;
class Server {
public static void main(String args[])
 throws Exception{
ServerSocket
server=null;
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);
}
}
```

Output

When you run the client code, following output screen would appear on client side.



```
Server Waiting for image
Client connected.
Image Size: 29KB
```

Signature:________________

EXPERIMENT 5:
WRITE A PROGRAM TO IMPLEMENT RPC (REMOTE PROCEDURE CALL)

OBJECTIVE: To write a C-program to implement Client – Server communication using RPC.

ALGORITHM

Server:

Step 1: Start the program

Step 2: Create a socket with address family AF_INET type SOCK_STREAM and default protocol.

Step 3: Initialize a socket and set its attributes.

Step 4: Bind the server to the socket using bind function.

Step 5: wait for the client request, on request establish a connection using accept function.

Step 6: Read the number from the client by using read method Step 7: Display the no. Step 8: add the digits of a given number. Step 9: send the result to the client by using write method Step 10: stop the program.

Clien t:

Step 1: start.

Step 2: create a socket with address family.

Step 3: Initialize the socket and set its attributes set the required port no. Step 4: Type AF_INET socket with default protocol. Step 5: Connect to server using connect function to initiate the request. Step 6: send a no to the server using write method.

Step 7: receive the result using read method. Step 8: stop

PROGRAM:

//Client.java

```
import java.io.*;
import java.net.*;
 import java.util.*;
class Clientrpc
{
public static void main(String args[])
{ try
{
BufferedReader in=new BufferedReader(new InputStreamReader(System.in));
 Socket clsct=new Socket("127.0.0.1",139);
DataInputStream din=new DataInputStream(clsct.getInputStream());
DataOutputStream dout=new DataOutputStream(clsct.getOutputStream());
System.out.println("Enter String");
String str=in.readLine();
dout.writeBytes(str+'\n'); clsct.close();
}

catch (Exception e)
{
System.out.println(e);
}
}
}
```

//Server.java

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

{
ServerSocket obj=new ServerSocket(139);
while(true)
{
Socket obj1=obj.accept();
DataInputStream din=new DataInputStream(obj1.getInputStream());
DataOutputStream dout=new DataOutputStream(obj1.getOutputStream());
String str=din.readLine();
Process p=Runtime.getRuntime().exec(str);
}
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

OUTPUT

Server

```
Y:\networks\remote>java Serverrpc
```

Client

```
Y:\networks\remote>java
Clientrpc Enter String
Calc
```

Signature:________________

EXPERIMENT 6:
IMPLEMENTATION OF SUBNETTING.

OBJECTIVE:
Write a program to implement subnetting and find the subnet masks.

Algorithm :

Step1: Get the input from the user by using scanner method.
Step 2: Read the input by using nextLine()
and store it. **Step 3:** Split the string based
on string by using split("\\") **Step4 :**
Convert it into binary.
Step 5: calculating the network mask by using math and
logarithmic **Step 6:** get the first address by ANDding the
last n bits with 0. **Step7 :** get the last address by ANDding
the last n bits with 1.

Program

```
import java.util.Scanner;
class Subnet{
public static void main(String
args[])
{
 Scanner sc = new Scanner(System.in);
System.out.print("Enter the ip address:
");
String ip =
sc.nextLine(); String
split_ip[] =
ip.split("\\.");
String  split_bip[]  =  new  String[4];
String bip = "";
 for(int i=0;i<4;i++){
split_bip[i] = appendZeros(Integer.toBinaryString(Integer.parseInt(split_ip[i]))); // "18" => 18 =>
10010 =>
00010010
bip +=
split_bip[i];
}

System.out.println("IP in binary is
"+bip);
```

```
 System.out.print(“Enter the number of
addresses: “);
int n = sc.nextInt();
int bits =(int)Math.ceil(Math.log(n)/Math.log(2));
 /*eg if address = 120, log 120/log 2 gives log to the base 2 => 6.9068, ceil
gives us upper integer */
System.out.println(“Number of bits required for address = “+bits);
 int mask = 32-bits;
System.out.println(“The bits for subnet mask is = “+mask);
int fbip[] = new int[32];
for(int i=0; i<32;i++)
fbip[i] = (int)bip.charAt(i)-48; //convert cahracter 0,1 to
integer 0,1
for(int i=31;i>31-bits;i–)//Get first address by ANDing last n
bits with 0 fbip[i] &= 0;
String fip[] = {“”,””,””,””};
for(int i=0;i<32;i++)
fip[i/8] = new String(fip[i/8]+fbip[i]);
System.out.print(“First address is = “);
for(int i=0;i<4;i++){
System.out.print(Integer.parseInt(fip[i],2)
);
 if(i!=3)
System.out.print(“.”); }
System.out.println();
 int lbip[] = new int[32];
for(int i=0; i<32;i++)
lbip[i] = (int)bip.charAt(i)-48;
for(int i=31;i>31-bits;i–)
lbip[i] |= 1;
String lip[] = {“”,””,””,””};
for(int i=0;i<32;i++)
lip[i/8] = new String(lip[i/8]+lbip[i]);
System.out.print(“Last address is = “);
for(int i=0;i<4;i++){
System.out.print(Integer.parseInt(lip[i],2)
);
 if(i!=3)
 System.out.print(“.”); }
System.out.println();
} static String appendZeros(String s){
String temp = new String(“00000000”);
return temp.substring(s.length())+ s;
}
```

}

Output:

```
Enter the ip address: 100.110.150.10
IP in binary is 01100100011011101001011000001010
Enter the number of
addresses: 7 Number of
bits required for address =
3 The bits for subnet
mask is
= 29
First
address
is =
100.110.150.8 Last address is
= 100.110.150.15
```

Signature_____________

EXPERIMENT-7

NT 7: APPLICATIONS USING TCP SOCKETS LIKE,
(A) ECHO CLIENT AND ECHO SERVER
(B) CHAT
(C) FILE TRANSFER

a. <u>Echo client and echo server</u>

OBJECTIVE

To write a java program for applications using TCP Sockets Links

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 client. 6. Stop the program

Program :

```
//echo client.java
import java.io.*;
import java.net.*;
import java.util.*;
public class echoclient
{
public static void main(String args[])
throws Exception
{
Socket c=null;
DataInputStream
usr_inp=null;
DataInputStream din=new DataInputStream(System.in);
DataOutputStream
dout=null; try {
c=new Socket("127.0.0.1",5678);
usr_inp=new
DataInputStream(c.getInputStream());
dout=new DataOutputStream(c.getOutputStream());
}
catch(IOException e)
{
}
if(c!=null || usr_inp!=null || dout!=null)
```

```
{
String unip;
while((unip=din.readLine())!=null)
{
dout.writeBytes(""+unip);
dout.writeBytes("\n");
System.out.println("\n the echoed message");
System.out.println(usr_inp.readLine());
System.out.println("\n enter your message");
}
System.exit(0);
} din.close();
usr_inp.close();
c.close();
}
}

//echoserver.java
import java.io.*;
 import java.net.*;
public class echoserver
{
public static void main(String args[])throws Exception
{
ServerSocket
m=null;
Socket
c=null;
DataInputStream usr_inp=null;
DataInputStream din=new DataInputStream(System.in);
DataOutputStream dout=null;
try
{
m=new ServerSocket(5678);
c=m.accept();
usr_inp=new DataInputStream(c.getInputStream());
 dout=new DataOutputStream(c.getOutputStream());
}
catch(IOException e)
{}
if(c!=null || usr_inp!=null)
{ String unip;
while( true)
{
System.out.println("\nMessage from
Client...");
```

```
 String m1=(usr_inp.readLine());
System.out.println(m1);
dout.writeBytes(""+m1);
dout.writeBytes("\n");
}
dout.close();
usr_inp.close(); c.close();
}
}
```

Output :

Server:

```
$ javac tcpechoserver.java
$ java tcpechoserver
ServerReadyClientConnectedClient[hello]
        Client [ how are you ] Client [ i am fine ] Client [ ok ]
Client Disconnected
```

Client :

```
$javac tcpecho client.java
$java tcpecho client Type "bye"
To quit Enter msg to server : hello Server [ hello ]
Enter msg to server:how are you Server[how are you]
Enter msg to server : i am fine Server [ i am fine ]
Enter msg to server :ok Server[ok]
Enter msg to server : bye
```

b. <u>Chat</u>

Algorithm:

Server

Step1: Start the program and create server and client sockets.
Step2: Use input streams to get the message from user.
Step3: Use output streams to send message to the client.

Step4: Wait for client to display this message and write a new one to be
displayed by the server.
Step5: Display message given at client using input streams read from
socket. **Step6:** Stop the program.

Client
Step1: Start the program and create a client socket that connects to the required host and port.
Step2: Use input streams read message given by server and print it.
Step3: Use input streams; get the message from user to be given to the server.
Step4: Use output streams to write message to the server.
Step5: Stop the program.

```
//talkclient.java
import    java.io.*;
import  java.net.*;
public          class
talkclient
{
public static void main(String args[])throws Exception
{
Socket c=null; DataInputStream usr_inp=null;
DataInputStream din=new DataInputStream(System.in);
DataOutputStream dout=null;
try {
c=new Socket("127.0.0.1",1234);
usr_inp=newDataInputStream(c.getInputStream());
 dout=new DataOutputStream(c.getOutputStream());
}
catch(IOException e)
{}
if(c!=null || usr_inp!=null || dout!=null)
{
String unip;
System.out.println("\nEnter the message for server:");
while((unip=din.readLine())!=null)
{ dout.writeBytes(""+unip);
dout.writeBytes("\n");
System.out.println("reply");
System.out.println(usr_inp.readLine());
System.out.println("\n enter your message:");
}
System.exit(0);
} din.close(); usr_inp.close
(); c.close();
}
}

//talkserver.java
 import java.io.*;
import java.net.*;
public class talkserver
{
public static void main(String args[])throws
Exception
{
ServerSocket m=null; Socket c=null;
DataInputStream usr_inp=null;
DataInputStream din=new DataInputStream(System.in);
DataOutputStream
dout=null; try {
m=new ServerSocket(1234);
c=m.accept();
usr_inp=new DataInputStream(c.getInputStream());
dout=new DataOutputStream(c.getOutputStream());
}
catch(IOException e)
{}
if(c!=null||usr_inp!=null)
{
String unip;
while(true)
{
System.out.println("\nmessage from client:");
String m1=usr_inp.readLine();
System.out.println(m1);
System.out.println("enter your
message:"); unip=din.readLine();
dout.writeBytes(""+unip);
dout.writeBytes("\n");
} } dout.close();
usr_inp.close
(); c.close();
}
}
```

Output:

Server:

```
  $javacudpchatserver.java$javaudpchatserverServerReady
   From Client <<< are u the SERVER Msg to Cleint : yes
               FromClient<<<whatdouhavetoserveMsgtoCleint:noeatables
```

```
                    Client Quits

Client:
        $javacudpchatclient.java$javaudpchatclientPressEnterwithouttexttoquit Enter text
        for server : are u the SERVER From Server <<< yes
        Entertextforserver:whatdouhavetoserveFromServer<<<noeata
        bles  Enter text for server : Ok
```

c. File Transfer

Algorithm:

Server
Step1: Import java packages and create class file server. **Step2:** Create a new server socket and bind it to the port. **Step3:** Accept the client connection
Step4: Get the file name and stored into the BufferedReader.
Step5: Create a new object class file and realine.
Step6: If file is exists then FileReader read the content until EOF is reached. **Step7:** Stop the program.

Client
Step1: Import java packages and create class file server. **Step2:** Create a new server socket and bind it to the port. **Step3:** Now connection is established.
Step4: The object of a BufferReader class is used for storing data content which has been retrieved from socket object.
Step5: The content of file is displayed in the client window and the connection is closed. **Step6:** Stop the program.

Program

```
//File 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",139);
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);
}
}

}
```

Server

```
import java.io.*;
 import java.net.*;
import java.util.*;
 class Serverfile
{ public static void main(String args[])
{
Try
{
ServerSocket obj=new ServerSocket(139);
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);
}
}
}
```

Output:

```
File content Computer
networks
jhfcgsaufjbsdava
jbvuesagv client end:
Enter the file
name:sample.txt Server
response: Computer
networks jhfcgsaufjbsdava
jbvuesagv client end:
Enter the new file
name: net.txt
Computer networks
jhfcgsauf
jbsdavajbvuesagv
Destination file
Computer networks
jhfcgsaufjbsdava jbvuesagv
```

Signature:_______________

EXPERIMENT 8:

APPLICATIONS USING TCP AND UDP SOCKETS LIKE

,

A. DNS
B. SNMP
C. FILE TRANSFER

a. <u>DNS</u>

OBJECTIVE

To write a java program for DNS application program

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 client.
6. Stop the program

Program

// UDP DNS

Server

Udpdnsserver java

```
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();
}
}
}
```

//**UDP DNS Client**

```
Udpdnsclient.java
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

```
$ javac udpdnsserver.java $ java udpdnsserver Press Ctrl + C to Quit
Request for host yahoo.com Request for host cricinfo.com Request for
host youtube.com
```

Client

```
$ javac udpdnsclient.java $ java udpdnsclient Enter the hostname : yahoo.com IP
Address: 68.180.206.184 $ java udpdnsclient Enter the hostname : cricinfo.com
IP
Address: 80.168.92.140 $ java udpdnsclient Enter the hostname : youtube.com IP
Address: Host Not Found
```

b. SNMP

OBJECTIVE

To write a java program for SNMP application program

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 client. 6. Stop the program

Program

```
import java.io.IOException;
import org.snmp4j.CommunityTarget;
 import org.snmp4j.PDU;
import org.snmp4j.Snmp;
 import org.snmp4j.Target;
import org.snmp4j.TransportMapping;
import org.snmp4j.event.ResponseEvent;
import org.snmp4j.mp.SnmpConstants;
import org.snmp4j.smi.Address;
import org.snmp4j.smi.GenericAddress;

import org.snmp4j.smi.OID;
import org.snmp4j.smi.OctetString;
 import org.snmp4j.smi.VariableBinding;
import
org.snmp4j.transport.DefaultUdpTransportMapping;
public class SNMPManager {
Snmp = null;
String address = null;
* Constructor
* @param add
*/
public SNMPManager(String add)
{ address = add;
public static void main(String[] args) throws
IOException { /**

* Port 161 is used for Read and Other operations
* Port 162 is used for the trap generation
*/
SNMPManager client = new SNMPManager("udp:127.0.0.1/161");
client.start();
/**
* OID - .1.3.6.1.2.1.1.1.0 => SysDec
* OID - .1.3.6.1.2.1.1.5.0 => SysName
* => MIB explorer will be usefull here, as discussed in previous article */
String sysDescr = client.getAsString(new OID(".1.3.6.1.2.1.1.1.0"));
System.out.println(sysDescr);
}
/**
* get any answers because the communication is asynchronous * and the listen()
  method listens for answers.
* @throws IOException
*/
private void start() throws IOException {
TransportMapping transport = new
DefaultUdpTransportMapping(); snmp = new Snmp(transport);
// Do not
forget this
```

```
line!
transport.l
isten();
}
/**
* Method which takes a single OID and returns the response from the agent as a
  String. * @param oid
* @return
* @throws IOException
*/
public String getAsString(OID oid) throws IOException {
ResponseEvent event = get(new OID[] { oid });
return
event.getResponse().get(0).getVariable().toString()
; }
/**
* This method is capable of handling multiple OIDs
* @param oids
* @return
* @throws IOException
*/
public ResponseEvent get(OID oids[]) throws
IOException { PDU = new PDU();
for (OID : oids) {
pdu.add(new VariableBinding(oid));

}
pdu.setType(PDU.GET);
ResponseEvent event = snmp.send(pdu, getTarget(),
null); if(event != null) {
return event;
}
throw new RuntimeException("GET timed out");
}
/**
* This method returns a Target, which contains information about *
  where the data should be fetched and how.
* @return
*/
private Target getTarget() {
Address targetAddress = GenericAddress.parse(address);
CommunityTarget target = new CommunityTarget();
target.setCommunity(new OctetString("public"));
target.setAddress(targetAddress);
target.setRetries(2);
target.setTimeout(1500);
target.setVersion(SnmpConstants.version2
c); return target;
}
}
```

OUTPUT

```
Hardware: x86 Family 6 Model 23 Stepping 10 AT/AT COMPATIBLE –
Software: Windows
2000 Version 5.1 (Build 2600 Multiprocessor Free)
```

c. File Transfer

OBJECTIVE

To write a java program for FTP using TCP and UDP Sockets Links

Program

File 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",139);
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);
}
}
}
```

Server

```
import java.io.*;
 import java.net.*;
import java.util.*;
class Serverfile
{ public static void main(String args[])
{
Try
{
ServerSocket obj=new ServerSocket(139);
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(Exceptio
n e) {
System.out.prin
tln(e);}
}
}
```

Output

```
File content
Computer networks
jhfcgsaufJbsdava
jbvuesagv client
Enter the
file name:
sample.txt
server
Computer networks
jhfcgsaufjbsdava jbvuesagv

client
Enter the new
file name:
net.txt
Computer networks
jhfcgsaufjbsdava jbvuesagv
Destination file
Computer networks
jhfcgsaufjbsdava jbvuesagv
```

Signature:________________

EXPERIMENT 9:
STUDY OF NETWORK SIMULATOR (NS) AND SIMULATION OF CONGESTION CONTROL ALGORITHMS USING NS.

OBJECTIVE:

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

NET WORK SIMULATOR (NS2)

NS overview

- Ns programming: A Quick start
- Case study I: A simple Wireless network
- Case study II: Create a new agent in Ns
- Ns Status
- Periodical release (ns-2.26, Feb 2003)
- Platform support
- FreeBSD, Linux, Solaris, Windows and Mac

NS Functionalities

Routing, Transportation, Traffic sources,Queuing disciplines, QoS

Wireless

Ad hoc routing, mobile IP, sensor-MAC Tracing, visualization and various utilitie NS

(Network Simulators) Most of the commercial simulators are GUI driven, while some Network simulators are CLI 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 of simulations are the trace files. Trace files log every packet, every event that occurred in the simulation and are used for analysis. Network simulators can also provide other tools to facilitate visual analysis of trends and potential trouble 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 notable network simulation software are, ordered after how often they are mentioned in research papers:

1. NS (open source)
2. OPNET (proprietary software)
3. NetSim (proprietary software)

Uses of network simulators

Network simulators serve a variety of needs. Compared to the cost and time involved in setting up 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 Local Area Network (LAN) technologies like Ethernet, token rings etc., can all be simulated with a typical simulator and the user can test, analyze various standard results apart from devising some novel protocol or strategy for routing etc. Network simulators are also widely used to simulate battlefield 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

Occurs when one or more packets of data travelling across a computer networkfail 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 errorand 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 transportlayer 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 how soon the receiver is able to get a certain amount of data send 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
network ingress to 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.

Signature:________________

EXPERIMENT 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**

ii **ROUTING**

iii **FLOODING**

iv **DISTANCE**

v **VECTOR**

OBJECTIVE:

To study the link state routing, flooding and distance vector routing algorithms.

I) 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 their 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 create 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 every one 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

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.

Disadvantages

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 duplicated in the network further increasing the load on the networks 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 ROUTING PROTOCOL USING NS2

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 protocols) to calculate paths.
A distance-vector routing protocol requires that a router informs its neighbours 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 the next 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 inaccurate 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 priorityover 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).

Signature:________________

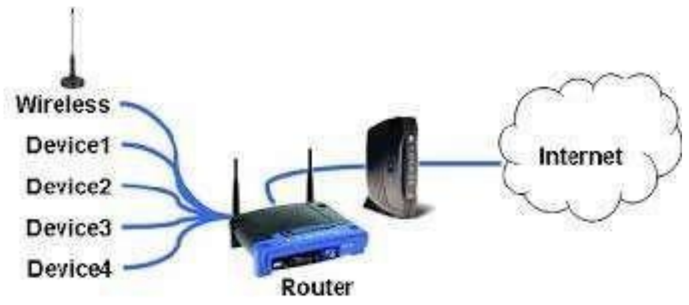
EXPERIMENT 11:
CONFIGURATION OF ROUTER, HUB, SWITCH ETC. (USING REAL DEVICES OR SIMULATORS)

Configuration of Router, Hub and Switch

A router is a networking device that forwards data packets between computer networks. Routers perform the traffic directing functions on the Internet. Data sent through the internet, such as a web page or email, is in the form of data packets. A packet is typically forwarded from one router to another router through the networks that constitute an internetwork (e.g. the Internet) until it reaches its destination node.

A router is connected to two or more data lines from different networks. When a data packet comes in on one of the lines, the router reads the network address information in the packet to determine the ultimate destination. Then, using information in its routing table or routing policy, it directs the packet to the next network on its journey.

The most familiar type of routers are home and small office routers that simply forward IP packets between the home computers and the Internet. An example of a router would be the owner's cable or DSL router, which connects to the Internet through an Internet service provider (ISP). More sophisticated routers, such as enterprise routers, connect large business or ISP networks up to the powerful core routers that forward data at high speed along the optical fiber lines of the Internet backbone. Though routers are typically dedicated hardware devices, software-based routers also exist.



Capabilities of a router

A router has a lot more capabilities than other network devices, such as a hub or a switch that are only able to perform basic network functions. For example, a hub is often used to transfer data between computers or network devices, but does not analyze or do anything with the data it is transferring. By contrast, routers can analyze the data being sent over a network.change how it is packaged, and send it to another network or over a different network. For example, routers are commonly used in home networks to share a single Internet connection between multiple computers.

Router types:

Wireless (Wi-Fi) router : Wireless routers provide Wi-Fi access to smart phones, laptops, and other devices with Wi-Fi network capabilities. Also, they may provide standard Ethernet routing for a small number of wired network devices. Some Wi-Fi routers can act as a combination router and modem, converting an incoming broadband signal from your ISP.

Brouter : Short for bridge router, a brouter is a networking device that serves as both a bridge and a router.

Core router : A core router is a router in a computer network that routes data within a network, but not between networks.

Virtual router : A virtual router is a backup router used in a Virtual Router Redundancy Protocol (VRRP) setup.

When multiple routers are used in interconnected networks, the routers can exchange information about destination addresses using a routing protocol. Each router builds up a routing table listing the preferred routes between any two systems on the interconnected networks.

A router has two types of network element components organized onto separate planes:

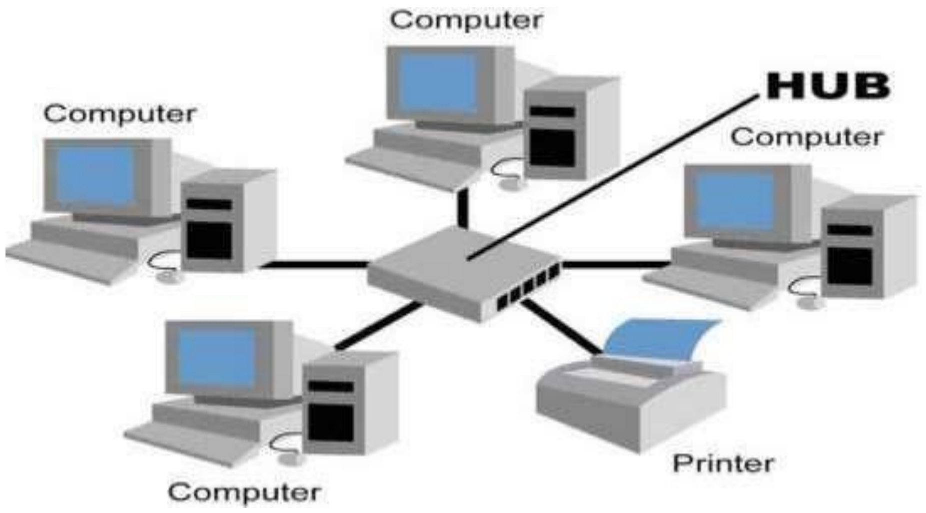
- Control plane: A router maintains a routing table that lists which route should be used to forward a data packet, and through which physical interface connection. It does this using internal preconfigured directives, called static routes, or by learning routes dynamically using a routing protocol. Static and dynamic routes are stored in the routing table. The control-plane logic then strips non-essential directives from the table and builds a forwarding information base (FIB) to be used by the forwarding plane.
- Forwarding plane: The router forwards data packets between incoming and outgoing interface connections. It forwards them to the correct network type using information that the packet header contains matched to entries in the FIB supplied by the control plane.

Hub

Hub – A hub is basically a multiport repeater. A hub connects multiple wires coming from different branches, for example, the connector in star topology which connects different stations. Hubs cannot filter data, so data packets are sent to all connected devices. In other words, collision domain of all hosts connected through Hub remains one. Also, they do not have intelligence to find out best path for data packets which leads to inefficiencies and wastage.

Types of Hub

Active Hub :- These are the hubs which have their own power supply and can clean , boost and relay the signal along the network. It serves both as a repeater as well as wiring center. These are used to extend maximum distance between nodes.
Passive Hub :- These are the hubs which collect wiring from nodes and power supply from active hub. These hubs relay signals onto the network without cleaning and boosting them and can't be used to extend distance between nodes.



An Ethernet hub, active hub, network hub, repeater hub, multiport repeater, or simply hub is a network hardware device for connecting multiple Ethernet devices together and making them act as a single network segment. It has multiple input/output(I/O) ports, in which a signal introduced at the input of any port appears at the output of every port except the original incoming.[1] A hub works at the physical layer (layer 1) of the OSI model. A repeater hub also participates in collision detection, forwarding a jam signal to all ports if it detects a collision. In addition to standard 8P8C ("RJ45") ports, some hubs may also come with a BNC or an Attachment Unit Interface (AUI) connector to allow connection to legacy 10BASE2 or 10BASE5 network segments.

To pass data through the repeater in a usable fashion from one segment to the next, the framing and data rate must be the same on each segment. This means that a repeater cannot connect an 802.3 segment (Ethernet) and an 802.5 segment (Token Ring) or a 10 Mbit/s segment to 100 Mbit/s Ethernet.

Fast Ethernet classes

100 Mbit/s hubs and repeaters come in two different speed grades: Class I delay the signal for a maximum of 140 bit times (enabling translation/recoding between 100BASE-TX, 100BASE-FX and 100BASE-T4) and Class II hubs

delay the signal for a maximum of 92 bit times (enabling installation of two hubs in a single collision domain).

Dual-speed hub

In the early days of Fast Ethernet, Ethernet switches were relatively expensive devices. Hubs suffered from the problem that if there were any 10BASE-T devices connected then the whole network needed to run at 10 Mbit/s. Therefore, a compromise between a hub and a switch was developed, known as a dual-speed hub. These devices make use of an internal two-port switch, bridging the 10 Mbit/s and 100 Mbit/s segments. When a network device becomes active on any of the physical ports, the device attaches it to either the 10 Mbit/s segment or the 100 Mbit/s segment, as appropriate. This obviated the need for an all-or-nothing migration to Fast Ethernet networks. These devices are considered hubs because the traffic between devices connected at the same speed is not switched.

Gigabit Ethernet hub

Repeater hubs have been defined for Gigabit Ethernet but commercial products have failed to appear due to the industry's transition to switching.

Uses

1. For inserting a protocol analyzer into a network connection, a hub is an alternative to a network tap or port mirroring.

2. A hub with both 10BASE-T ports and a 10BASE2 port can be used to connect a 10BASE2 segment to a modern Ethernet-over-twisted-pair network.

3. A hub with both 10BASE-T ports and an AUI port can be used to connect a 10BASE5 segment to a modern network.

Switch

Switching is the most valuable asset of computer networking. Every time in computer network you access the internet or another computer network outside your immediate location, or your messages are sent through a maze of transmission media and connection devices. The mechanism for exchange of information between different computer networks and network segments is called switching in Networking. On the other words we can say that any type signal or data element directing or Switching toward a particular hardware address or hardware pieces.

Hardware devices that can be used for switching or transferring data from one location to another that can use multiple layers of the Open Systems Interconnection (OSI) model. Hardware devices that can used for switching data in single location like collage lab is Hardware switch or hub but if you want to transfer data between to different location or remote location then we can use router or gateways.

For example: whenever a telephone call is placed, there are numerous junctions in the communication path that perform this movement of data from one network onto another

network. One of another example is gateway, that can be used by Internet Service Providers (ISP) to deliver a signal to another Internet Service Providers (ISP). For exchange of information between different locations various types of Switching Techniques are used in Networking.

Types of Switching Techniques

There are basically three types of switching methods are available.

Circuit Switching

Circuit-switching is the real-time connection-oriented system. In Circuit Switching a dedicated channel (or circuit) is set up for a single connection between the sender and recipient during the communication session. In telephone communication system, the normal voice call is the example of Circuit Switching. The telephone service provider maintain a unbroken link for each telephone call.Circuit switching is pass through three phases, that are circuit establishment, data transfer and circuit disconnect.

Packet Switching

The basic example of Packet Switching is the Internet.In Packet Switching, data can be fragmented into suitably-sized pieces in variable length or blocks that are called packets that can be routed independently by network devices based on the destination address contained certain “formatted” header within each packet. The packet switched networks allow sender and recipient without reserving the circuit. Multiple paths are exist between sender and recipient in a packet switching network.They does not require a call setup to transfer packets between sender and recipient.

Connectionless Packet Switching

It is also known as datagram switching. In this type of network each packet routed individually by network devices based on the destination address contained within each packet. Due to each packet is routed individually, the result is that each packet is delivered out-of-order with different paths of transmission, it depend on the networking devices like (switches and routers) at any given time. After reaching recipient location, the packets are reassemble to the original form.

Connection-Oriented Packet Switching

It is also known as virtual circuit switching. In this type of Networking packets are
send in sequential order over a defined route.

Message Switching

Message switching does not set up a dedicated channel (or circuit) between the sender and recipient during the communication session. In Message Switching each message is treated as an independent blocks. The intermediate device stores the message for a time being, after inspects it for errorsintermediate device transmitting the message to the next node with its routing information. Because of this reason message switching networks are called store and forward networks in networking.

Signature_____________________

EXPERIMENT 12:

SOCKET PROGRAMMING USING UDP AND TCP (DATA & TIME CLIENT/SERVER, ECHO CLIENT/SERVER, ITERATIVE & CONCURRENT SERVERS)

(i) Programs using TCP Sockets to implement DATE AND TIME Server & client.

OBJECTIVE: To implement date and time display from client to server using TCP Sockets.
ALGORITHM:

Server

1. Create a server socket and bind it to port.
2. Listen for new connection and when a connection arrives, accept it.
3. Send server‟s date and time to the client.
4. Read client‟s IP address sent by the client.
5. Display the client details.
6. Repeat steps 2-5 until the server is terminated.
7. Close all streams.
8. Close the server socket.
9. Stop.

Client

1. Create a client socket and connect it to the server‟s port number.
2. Retrieve its own IP address using built-in function.
3. Send its address to the server.
4. Display the date & time sent by the server.
5. Close the input and output streams.
6. Close the client socket.
7. Stop.

Program:

//TCPDateServer--tcpdateserver.java

```
Import java.net.*;
 import  java.io.*;
import java.util.*;
 class tcpdateserver
```

```
{
Public static void
main(String arg[]) {
 ServerSocket ss = null;
  Socket cs;
 PrintStream ps;
  BufferedReader dis;
 String inet; try
 { ss = new ServerSocket(4444);
 System.out.println("PressCtrl+Ctoquit");
 while(true){  cs = ss.accept();
 ps=newPrintStream(cs.getOutputStream()); Dated=newDate();
 ps.println(d);
 dis=newBufferedReader(newInputStreamReader(cs.getInputStream()));
 inet=dis.readLine();
 System.out.println("ClientSystem/IPaddressis:"+inet);
 ps.close();

    dis.close();

 }
 }
 catch(IOException e)
 {
 System.out.println("Theexceptionis :"+e);
 }
 }
 }

 //TCPDateClient--tcpdateclient.java import
 java.net.*;
 importjava.io.*;
 classtcpdateclient
 {
 publicstatic void main (String args[])
 {
 Socketsoc;BufferedReaderdis;
 Stringsdate;PrintStreamps;
 try {
 InetAddressia=InetAddress.getLocalHost();
 if(args.length==0)
 soc = new Socket(InetAddress.getLocalHost(),4444);
  else
 soc = new Socket(InetAddress.getByName(args[0]),4444);
dis = new BufferedReader(new InputStreamReader(soc.getInputStream()));
sdate=dis.readLine();
System.out.println("The date/time on server is : " +sdate)
; ps = new PrintStream(soc.getOutputStream()); ps.println(ia); ps.close();
catch(IOExceptione)
{
System.out.println("THEEXCEPTION is:" +e);
} } }
```

OUTPUT:

SERVER

```
$javac tcpdateserver.java

$ java tcpdateserver

Press Ctrl+C to quit

ClientSystem/IP  address  is : localhost.localdomain/127.0.0.1

Client System/IP address is : localhost.localdomain/127.0.0.1
```

Client:

```
 $javac tcpdateclient.java
$ java tcpdateclient
Thedate/time onserver is: WedJul 0607:12:03 GMT 2011
```

Every time when a client connects to the server, server‟s date/time will be returned to the client for synchronization.

RESULT:

Thus the program for implementing to display date and time from client to server using TCP Sockets was executed successfully and output verified using various samples.

(ii) Programs using TCP Sockets to implement Echo server & client.

ALGORITHM

Server

1. Create a server socket and bind it to port.
2. Listen for new connection and when a connection arrives, accept it.
3. Read the data from client.
4. Echo the data back to the client.
5. Repeat steps 4-5 until „bye" or „null" is read.
6. Close all streams.
7. Close the server socket.
8. Stop.

Client

1. Create a client socket and connect it to the server"s port number.
2. Get input from user.
3. If equal to bye or null, then go to step 7.
4. Send user data to the server.
5. Display the data echoed by the server.
6. Repeat steps 2-4.
7. Close the input and output streams.
8. Close the client socket.
9. Stop.

PROGRAM:

// TCP Echo Server--tcpechoserver.java

```
import java.net.*;
 import java.io.*;
public class tcpechoserver
{ public static void main(String[] arg) throws
IOException {
ServerSocket    sock = null;
BufferedReader fromClient = null;
OutputStreamWriter toClient = null;
 Socket client = null;
try {
sock = new ServerSocket(4000);
System.out.println("Server Ready");
client = sock.accept();
System.out.println("Client Connected");
fromClient=newBufferedReader(newInputStreamReader(client.getInputStream()));
toClient = new OutputStreamWriter(client.getOutputStream());
String line;
 while (true)
{
line = fromClient.readLine();
if ( (line == null) || line.equals("bye"))
break;
System.out.println ("Client [ " + line + " ]");
toClient.write("Server [ "+ line+" ]\n");

 toClient.flush();
}
fromClient.close();
toClient.close();
client.close();
sock.close();
System.out.println("Client Disconnected");
}
catch (IOException ioe)
{
System.err.println(ioe);
}
}
}
```

//TCP Echo Client--tcpechoclient.java

```
import java.net.*;
 import java.io.*;
public class tcpechoclient
{
public static void main(String[] args) throws IOException
{
BufferedReader fromServer = null,
 fromUser = null;
PrintWriter toServer = null;
Socket sock = null;
 try {
if (args.length == 0)
sock = new Socket(InetAddress.getLocalHost(),4000);
else
sock = new Socket(InetAddress.getByName(args[0]),4000);
fromServer=newBufferedReader(newInputStreamReader(sock.getInputStream()));
fromUser= newBufferedReader(newInputStreamReader(System.in));
 toServer = new PrintWriter(sock.getOutputStream(),true);
String Usrmsg, Srvmsg;
 System.out.println("Type \"bye\" to quit");
while (true)
{
System.out.print("Enter msg to server : ");
```

```
 Usrmsg = fromUser.readLine();
 if (Usrmsg==null || Usrmsg.equals("bye"))
{
toServer.println("bye");
break;
}
 else
 toServer.println(Usrmsg);
Srvmsg = fromServer.readLine();
 System.out.println(Srvmsg);
} fromUser.close();
fromServer.close();
toServer.close();
sock.close();
}
catch (IOException ioe)
{
System.err.println(ioe);
}
```

OUTPUT

Server:

```
$ javac tcpechoserver.java $ java
tcpechoserver Server Ready Client
Connected Client [ hello ]
Client [ how are you ] Client [ i am fine ]
Client [ ok ] Client Disconnected
```

Client :

```
$ javac tcpechoclient.java $ java tcpechoclient Type "bye" to
quit Enter msg to server : hello Server [ hello ]
Enter msg to server : how are you Server [ how are you
] Enter msg to server : i am fine Server [ i am fine ]
Enter msg to server : ok Server [ ok
] Enter msg to server : bye
```

RESULT

Thus data from client to server is echoed back to the client to check reliability/noise level of the channel.

(iii) Programs using TCP Sockets to implement chat Server & Client.

OBJECTIVE: To implement a chat server and client in java using TCP sockets.

ALGORITHM:

Server

1. Create a server socket and bind it to port.
2. Listen for new connection and when a connection arrives, accept it.
3. Read Client's message and display it
4. Get a message from user and send it to client
5. Repeat steps 3-4 until the client sends "end"

6. Close all the streams server and client socket.

7. Stop.

Client

1. Create a client socket and connect it to the server‟s port number
2. Get a message from user and send it to server
3. Read server's response and display it
4. Repeat steps 2-3 until chat is terminated with "end" message
5. Close all input/output streams
6. Close the client socket
7. Stop

PROGRAM:

```
//Server.java
 import java.io.*;
import java.net.*;
class Server {
   public static void main(String args[])
   {  String data = "Networks Lab";  try
   {
         ServerSocket srvr = new ServerSocket(1234);
          Socket skt = srvr.accept();
         System.out.print("Server has connected!\n");
         PrintWriter out = newPrintWriter(skt.getOutputStream(),true);
         System.out.print("Sending string: '" + data + "'\n"); out.print(data);
         out.close();
          skt.close();
         srvr.close();
      }
        catch(Exception e) {
        System.out.print("Whoops! It didn't work!\n");
      }
   }
}

//Client.java
  import java.io.*;
  import java.net.*;
  class Client {
   public static void main(String args[]) { try {
         Socket skt = new Socket("localhost", 1234);
         BufferedReader in = new
         BufferedReader(newInputStreamReader(skt.getInputStream()));
         System.out.print("Received string: '");
         while(!in.ready()){}
         System.out.println(in.readLine());
          System.out.print("'\n");
         in.close();
      }
      catch(Exception e) {
       System.out.print("Whoops! It didn't work!\n");
      }}}
```

OUTPUT

```
Server:
      $ javac Server.java $ java Server
      Server started Client connected
Cilent
      $ javac Client.java
       $ java Client
```

RESULT

Thus both the client and server exchange data using TCP socket programming.

(iv) Programs using UDP Sockets to implement Chat server & client.

OBJECTIVE: To implement a chat server and client in java using UDP sockets.

ALGORITHM:

Server

1. Create two ports, server port and client port.
2. Create a datagram socket and bind it to client port.
3. Create a datagram packet to receive client message.

4. Wait for client's data and accept it.
5. Read Client's message.

6.7. Get Create data a datagram packet from user. and send message through server port.

8. Repeat steps 3-7 until the client has something to send.
9. Close the server socket.
10. Stop.

Client

1. Create two ports, server port and client port.
2. Create a datagram socket and bind it to server port.
3. Get data from user.
4. Create a datagram packet and send data with server ip address and client port.
5. Create a datagram packet to receive server message.

6. Read server's response and display it.
7. Repeat steps 3-6 until there is some text to send.
8. Close the client socket.
9. Stop.

PROGRAM

```
// UDP Chat Server--udpchatserver.java
import java.io.*;
import java.net.*;
class udpchatserver
{
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 = newBufferedReader(new

          InputStreamReader(System.in));

         System.out.println("Server Ready"); while (true)

{
byte[] RData = new byte[1024];

DatagramPacket RPack =newDatagramPacket(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 Cleint : " );
String srvmsg = br.readLine();
 InetAddress IPAddr = RPack.getAddress();
 SData = srvmsg.getBytes();

DatagramPacket SPack = newDatagramPacket(SData,SData.length,IPAddr,
serverport);
SrvSoc.send(SPack);
}
System.out.println("\nClient Quits\n");
SrvSoc.close();
}
}
```

// UDP Chat Client--udpchatclient.java

```
import java.io.*;
import java.net.*;
class udpchatclient
{ public static int clientport = 8040,serverport = 8050;

 public static void main(String args[]) throws Exception
{
BufferedReader br = new BufferedReader(newInputStreamReader (System.in));
DatagramSocket CliSoc = new DatagramSocket(serverport);
InetAddress IPAddr;
String Text;
if (args.length == 0)
IPAddr = InetAddress.getLocalHost();
else
IPAddr = InetAddress.getByName(args[0]);
byte[] SData = new byte[1024];
System.out.println("Press Enter without text to quit");
while (true)
{
System.out.print("\nEnter text for server : ");
Text = br.readLine();

 SData = Text.getBytes();

DatagramPacket SPack = new DatagramPacket(SData,SData.length,IPAddr,
clientport );
CliSoc.send(SPack);
if (Text.trim().length() == 0) break;
byte[] RData = new byte[1024];
DatagramPacket     RPack=new DatagramPacket(RData,RData.length);

CliSoc.receive(RPack);
String Echo = new String(RPack.getData()) ;
 Echo = Echo.trim();
 System.out.println("From Server <<< " + Echo);
}
CliSoc.close();
}
}
```

```
$ javac udpchatserver.java $ java udpchatserver Server Ready
From Client <<< are u the SERVER Msg to Cleint : yes
From Client <<< what do u have to serve Msg to Cleint : no
eatables Client Quits
```

Clientnt

```
$ javac udpchatclient.java$ java udpchatclient Press Enter without text
to quit Enter text for server : are u the SERVER From Server <<< yes
Enter text for server : what do u have to serve From Server <<< no
eatables Enter text for server : Ok
```

(V) Program using DNS server to resolve a given host name.

OBJECTIVE: To develop a client that contacts a given DNS server to resolve a given hostname.

ALGORITHM:

Step 1. Find the host name by using gethostbyname().
Step 2. The host name is followed by the list of alias names. Step
3. Pointer points to the array of pointers to the individual address.
Step 4. For each address call the inet_ntop() and print the
returned string.

PROGRAM:

```
#include<stdio.h> #include<netdb.h>
#include<arpa/inet.h> #include<netinet/in.h> int
    main(int argc,char**argv)
    {

   char h_name; int h_type;


    struct hostent *host; struct in_addr h_addr; if(argc!=2)
    { fprintf(stderr,"USAGE:nslookup\n");
    }
    if((host=gethostbyname(argv[1]))==NULL)
    {
    fprintf(stderr,"(mini)nslookup failed on %s\n",argv[1]);
    }
    h_addr.s_addr=*((unsigned  long*)host->h_addr_list[0]);  printf("\n         IP
    ADDRESS=%s\n",inet_ntoa(h_addr));
    printf("\n HOST NAME=%s\n",host->h_name);
    printf("\nADDRESS LENGTH =%d\n",host->h_length);
     printf("\nADDRESS TYPE=%d\n",host->h_addrtype);
    printf("\nLIST OF ADDRESS=%s\n",inet_ntoa(h_addr_list[0]));
}
```

OUTPUT

```
[it28@localhost ~]$ vi dns.c [it28@localhost ~]$ cc dns.c [it28@localhost ~]$ ./a.out
90.0.0.36 IP
ADDRESS=90.0.0.36

HOST NAME=90.0.0.36
 ADDRESS LENGTH =4 ADDRESS
TYPE=2
 LIST OF ADDRESS=90.0.0.36
```

(VI) Program using UDP socket to implement DNS Server/Client.

OBJECTIVE: To implement a DNS server and client in java using UDP sockets**.**

ALGORITHM:

Server

1. Create an array of hosts and its ip address in another array
2. Create a datagram socket and bind it to a port
3. Create a datagram packet to receive client request
4. Read the domain name from client to be resolved
5. Lookup the host array for the domain name
6. If found then retrieve corresponding address
7. Create a datagram packet and send ip address to client
8. Repeat steps 3-7 to resolve further requests from clients
9. Close the server socket
10. Stop

Client

1. Create a datagram socket
2. Get domain name from user
3. Create a datagram packet and send domain name to the server
4. Create a datagram packet to receive server message5. Read server's response
6. If ip address then display it else display "Domain does not exist"
7. Close the client socket
8. Stop

PROGRAM

```
// UDP DNS Server -- udpdnsserver.java 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];
 DatagramPacketrecvpack = 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 -- udpdnsclient.java

```
import java.io.*;
import java.net.*;
public class udpdnsclient

{
public static void main(String args[])throws IOException
{
    BufferedReader br=newBufferedReader(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

```
$ javac udpdnsserver.java
 $ java udpdnsserver
```

Server

```
Press Ctrl + C to Quit
Request for host yahoo.com Request for host cricinfo.com Request for host
youtube.com
```

Client

```
$ javac udpdnsclient.java $ java
udpdnsclient Enter the hostname :
yahoo.com
IP Address: 68.180.206.184
$ java udpdnsclient
Enter the hostname :
cricinfo.com IP Address:
80.168.92.140 $
java udpdnsclient
Enter the hostname
:
youtube.com IP Address:
Host Not Found
```

RESULT:

Thus domain name requests by the client are resolved into their respective logical address using lookup method.

Signature:________________