PRANVEER SINGH INSTITUTE OF TECHNOLOGY, KANPUR

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Even Semester 2023-24

B. Tech.- Third Year

Semester- VI

Lab File

COMPUTER NETWORKS

(KCS653)

Submitted To :

Faculty Name : Mr Gaurav Bajpai

Designation : Assistant Professor

Submitted By :

Name : Aakarshit Srivastava

Roll No. : 2101641520001

Section : CS-AI-3A

Table of Contents

Vision Statement of the Institute

To achieve excellence in professional education and create an ecosystem for the holistic development of all stakeholders.

Mission Statement of the Institute

To provide an environment of effective learning and innovation transforming students into dynamic, responsible and productive professionals in their respective fields, who are capable of adapting to the changing needs of the industry and society.

Vision Statement of the Department

To be a recognized department of Computer Science & Engineering that produces versatile computer engineers, capable of adapting to the changing needs of computer and related industry.

Mission Statements of the Department

1. To provide broad based quality education with knowledge and attitude to succeed in Computer Science & Engineering careers.
2. To prepare students for emerging trends in computer and related industry.
3. To develop competence in students by providing them skills and aptitude to foster culture of continuous and life-long learning.
4. To develop practicing engineers who investigate research, design and find workable solutions to complex engineering problems with awareness & concern for society as well as environment.

Program Educational Objectives (PEOs)

PEOs	Description
PEO1	The graduates will be efficient leading professionals with the knowledge of Computer Science & Engineering discipline that enables them to pursue higher education and/or successful careers in various domains.
PEO2	Graduates will possess capability of designing successful innovative solutions to real life problems that are technically sound, economically viable and socially acceptable.
PEO3	Graduates will be competent team leaders, effective communicators and capable of working in multidisciplinary teams following ethical values.
PEO4	The graduates will be capable of adapting to new technologies/tools, constantly upgrading their knowledge and skills with an attitude for lifelong learning.

Program Outcomes (POs)

POs	Graduate Attributes	Description
PO1	**Engineering Knowledge**	Apply the knowledge of mathematics, science and Computer Science & Engineering fundamentals to the solution of complex engineering problems.
PO2	**Problem Analysis**	Identify, formulate, review research literature, and analyze complex Computer Science & Engineering problems reaching substantiated conclusions using principles of mathematics, natural sciences, and engineering
PO3	**Design/Development of Solutions**	Design solutions for Computer Science & Engineering and allied fields related complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	**Investigation**	Use research-based knowledge of Computer Science & Engineering and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	**Modern Tool Usage**	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex Computer Science & Engineering activities with an understanding of the limitations.
PO6	**The Engineering and Society**	Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice in the field of Computer Science & Engineering.
PO7	**Environment and Sustainability**	Understand the impact of the professional Computer Science & Engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development

PO8	**Ethics**	Apply ethical principles and commit to professional ethics and responsibilities and norms of the Computer Science & Engineering practice
PO9	**Individual and Team Work**	Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	**Communication**	Communicate effectively on complex Computer Science & Engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	**Project Management and Finance**	Demonstrate knowledge and understanding of the Computer Science & Engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	**Life-long Learning**	Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSOs)

PSOs	Description
PSO1	Use algorithms, data structures/management, software design, concepts of programming languages and computer organization and architecture.
PSO2	Understand the processes that support the delivery and management of information systems within a specific application environment.

Evaluation Scheme and Guidelines

3rd Year VI Semester

SEMESTER- VI													
Sl. No.	Subject Codes	Subject	Periods			Evaluation Scheme				End Semester		Total	Credit
			L	T	P	CT	TA	Total	PS	TE	PE		
1	KCS601	Software Engineering	3	1	0	30	20	50		100		150	4
2	KCS602	Web Technology	3	1	0	30	20	50		100		150	4
3	KCS603	Computer Networks	3	1	0	30	20	50		100		150	4
4		Departmental Elective-III	3	0	0	30	20	50		100		150	3
5		Open Elective-I [Annexure - B(iv)]	3	0	0	30	20	50		100		150	3
6	KCS651	Software Engineering Lab	0	0	2				25		25	50	1
7	KCS652	Web Technology Lab	0	0	2				25		25	50	1
8	KCS653	Computer Networks Lab	0	0	2				25		25	50	1
9	NC+	Essence of Indian Traditional Knowledge/Constitution of India	2	0	0	15	10	25		50			
10		MOOCs (Essential for Hons. Degree)											
		Total	**0**	**3**	**6**							**900**	**21**

CT: Class Test
TA: Teacher Assessment
L-T-P: Lecture - Tutorial - Practical

Evaluation Scheme		**Marks**	**Sub-Total**
Internal	Performance	10	25
	Viva	5	
	Lab Record	5	
	Attendance	5	
External	University Exam	25	25
Grand Total			**50**

Syllabus

Following table outline the syllabus for Computer Networks Lab (KCS-653) as prescribed by ***Dr. A.P.J. Abdul Kalam Technical University, Uttar Pradesh, Lucknow***. The Syllabus can also seen on the university website: https://aktu.ac.in/pdf/syllabus/syllabus2021/B.Tech_CSE%20and%20CS%20Syllabus%20of%203rd%20Year%209%20March%202021.pdf

DETAILED SYLLABUS
1. Implementation of Stop and Wait Protocol and Sliding Window Protocol. 2. Study of Socket Programming and Client – Server model 3. Write a code simulating ARP /RARP protocols. 4. Write a code simulating PING and TRACEROUTE commands 5. Create a socket for HTTP for web page upload and download. 6. Write a program to implement RPC (Remote Procedure Call) 7. Implementation of Subnetting . 8. Applications using TCP Sockets like a. Echo client and echo server b. Chat c. File Transfer 9. Applications using TCP and UDP Sockets like d. DNS e. SNMP f. File Transfer 10. Study of Network simulator (NS).and Simulation of Congestion Control Algorithms using NS 11. 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 12. To learn handling and configuration of networking hardware like RJ-45 connector, CAT-6 cable, crimping tool, etc. 13. Configuration of router, hub, switch etc. (using real devices or simulators) 14. Running and using services/commands like ping, traceroute, nslookup, arp, telnet, ftp, etc. 15.Network packet analysis using tools like Wireshark, tcpdump, etc. 16. Network simulation using tools like Cisco Packet Tracer, NetSim, OMNeT++, NS2, NS3, etc. 17.Socket programming using UDP and TCP (e.g., simple DNS, data & time client/server, echo client/server, iterative & concurrent servers)
Note: The Instructor may add/delete/modify/tune experiments, wherever he/she feels in a justified manner **It is also suggested that open source tools should be preferred to conduct the lab (C , C++ , Java , NS3, Mininet, Opnet, TCP Dump, Wireshark etc.**

LAB PLAN

SUBJECT NAME: **COMPUTER NETWORKS LAB**

SUBJECT CODE: **KCS-653**

Lab Schedule: As per the time table

i) Course Objective:
The objective of this lab is to give the idea about phases of compiler and analyze how the code generation & optimization works in a translator.

ii) Course Outcomes

*Level of Bloom's Taxonomy	Level to be met	*Level of Bloom's Taxonomy	Level to be Met
L1: Knowledge	1	L2: Comprehension	2
L3: Application	3	L4: Analysis	4
L5: Evaluate	5	L6: Create	6

CO Number	Course Outcomes
KCS-653.1	Illustrate **[L2: Comprehension]** conceptual terms and various components and devices of computer networks.
KCS-653.2	Apply **[L3: Application]** the knowledge of Networks for configuring the performance of the computer network.

CO-PO & CO-PSO Correlation Matrix

Mapping of Course Outcomes with Program Outcomes and Program Specific Outcomes

	Program Outcomes												PSO	
COs	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**1**	**2**
KCS-653.1	3	3	-	-	-	-	-	-	-	-	-	-	-	-
KCS-653.2	-	3	-	-	3	-	-	-	-	-	-	-	3	-
AVG	3	3	-	-	3	-	-	-	-	-	-	-	3	-

Justification of CO-PO Mapping

KCS-653.1 with PO1 and PO2	**PO1:** KCS-653.1 is substantially mapped with PO1 as the students will understand the different components of Networks and the tools involved with the knowledge they possessed during the undergraduate programme. **PO2:** KCS-653.1 is substantially mapped with PO2 as the students will be able to analyze the complex problems of Computer Networks and may approach with innovative solutions with the knowledge they possessed during the undergraduate programme.
KCS-653.2 with PO2 and PO5	**PO2:** KCS-653.2 is substantially mapped with PO2 as the students will be able to configure the performance of the Network with the analysis they have done. **PO5:** KCS-653.2 is substantially mapped with PO5 as the students will use the modern tools to configure and evaluate the performance of a network.

Justification of CO-PSO Mapping

KCS-653.2 with PSO1	**PSO1:** KCS-653.2 is substantially mapped with PSO1 as the students will use the knowledge obtained in the undergraduate programme to maintain and configure the network.

List of Experiments as per AKTU

Sr.No.	Objectives	Date of Experiment	Date of Submission	Remark	Sign
1	Implementation of Stop and Wait Protocol and Sliding Window Protocol.				
2	Study of Socket Programming and Client – Server model				
3	Write a code simulating ARP /RARP protocols.				
4	Write a code simulating PING and TRACEROUTE commands				
5	Create a socket for HTTP for web page upload and download.				
6	Write a program to implement RPC (Remote Procedure Call)				
7	Implementation of Subnetting.				
8	Applications using TCP Sockets like a.Echo client and echo server b. Chat c. File Transfer				
9	Applications using TCP and UDP Sockets like a. DNS b. SNMP c. File Transfer				
10	Study of Network simulator (NS).and Simulation of Congestion Control Algorithms using NS				
11	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				
12	To learn handling and configuration of networking hardware like RJ-45 connector, CAT-6 cable, crimping tool, etc.				
13	Configuration of router, hub, switch etc. (using real devices or simulators)				
14	Running and using services/commands like ping, traceroute, nslookup, arp, telnet, ftp, etc.				
15	Network packet analysis using tools like Wireshark, tcpdump, etc.				
16	Network simulation using tools like Cisco Packet Tracer, NetSim, OMNeT++, NS2, NS3, etc.				
17	Socket programming using UDP and TCP (e.g., simple DNS, data & time client/server, echo client/server, iterative & concurrent servers)				
	Experiment beyond syllabus				
1	Programming using raw sockets				

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.

Problem Statement:

- A program to evaluate the reliability of stop and wait protocol.

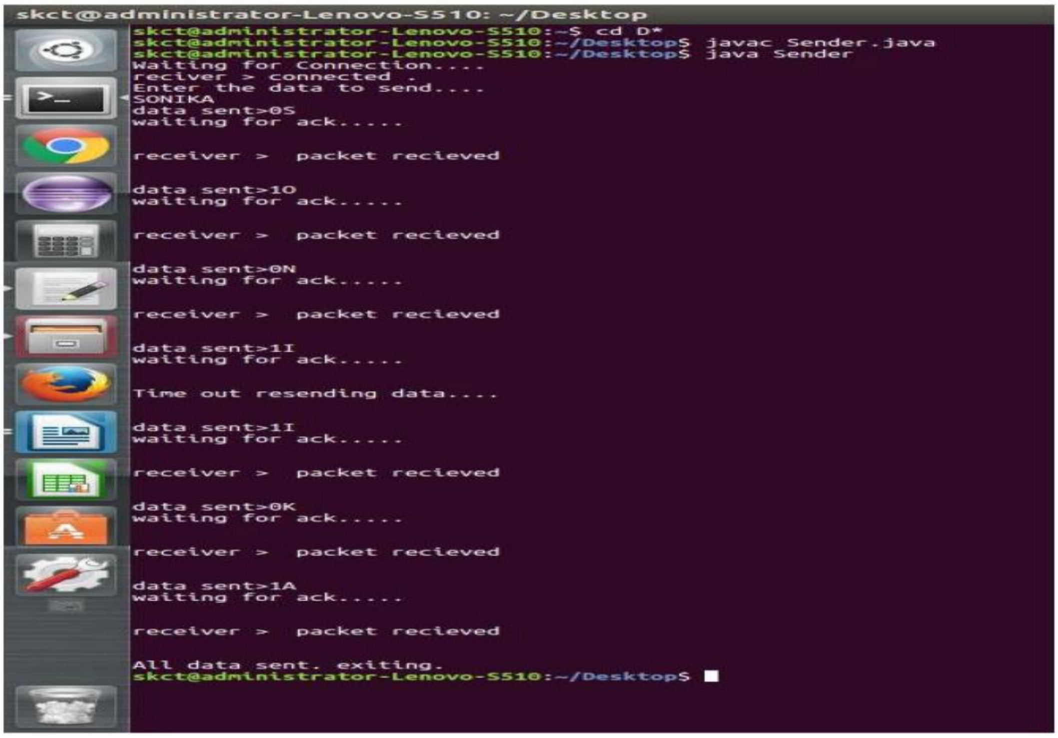
Sample Code:

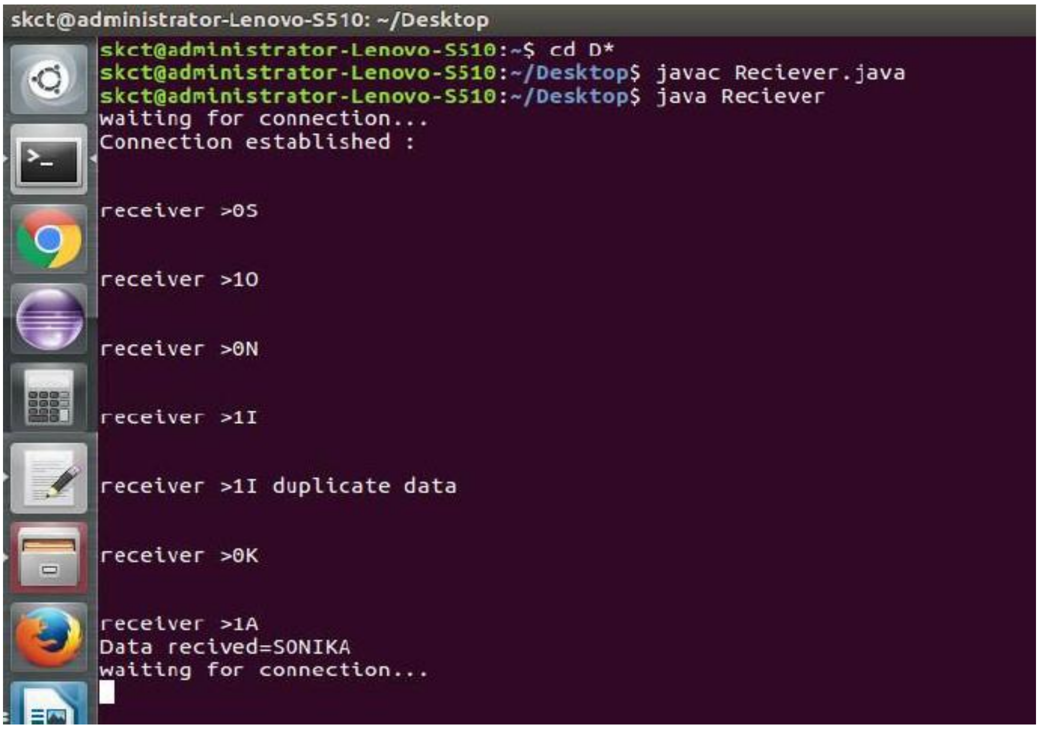
Sender.java
```
//import java packages
import java.io.*;
import java.net.*;
/*…System.in is an InputStream, we create an InputStreamReader which reads bytes
from System.in…*/
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
System.out.println("Waiting for Connection. .. ");
/*… Register service on port 2004…*/
sender = new Socket("localhost",2004);
sequence=0;
/*… Get a communication stream associated with the socket…*/
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. ..");
/*…readline() method read a line of text…*/
packet=br.readLine();
n=packet.length();
do{try{ if(i<n){
msg=String.valueOf(sequence);
msg=msg.concat(packet.substring(i,i+1));}
else if(i==n){ msg="end";
/*…method writeObject is used to write an object to the stream…*/
out.writeObject(msg);break;}
out.writeObject(msg);
sequence=(sequence==0)?1:0;
/*… method flushes this output stream and forces any buffered output bytes to be
written out…*/
out.flush();
```

Reciever.java
```
ServerSocketreciever;
Socket connection=null;
ObjectOutputStream out;
```

```
ObjectInputStream in;
public void run(){
/*… It reads bytes and decodes them into characters using a specified charset …*/
try{ BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
/*… Open your connection to a server, at port 2004…*/
reciever = new ServerSocket(2004,10);
System.out.println("waiting for connection...");
/*… Wait and accept a connection…*/
connection=reciever.accept();
sequence=0;
System.out.println("Connection established :");
/* …getOutputStream()-This method is used to take the permission to read data from client
system by the server or from the server system by the client…*/
out=new ObjectOutputStream(connection.getOutputStream());
out.flush();
/*…getInputStream()-This method take the permission to write the data from client program to
server program and server program to client program…*/
in=new ObjectInputStream(connection.getInputStream());
out.writeObject("connected .");
do{
try{
packet=(String)in.readObject();
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){
/*… The method writeObject is used to write an object to the stream …*/
out.writeObject(String.valueOf(sequence)); i++; }
elseout.writeObject(String.valueOf((sequence+1)%2));
```

Output:





(II) SLIDING WINDOW PROTOCOL

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

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.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();
}
}
```

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.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:

//SENDER OUTPUT

```
Enter the no. of frames : 4
Enter 4 Messages to be send
hiii
how r  u
i am fine
how is evryone
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
```

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.

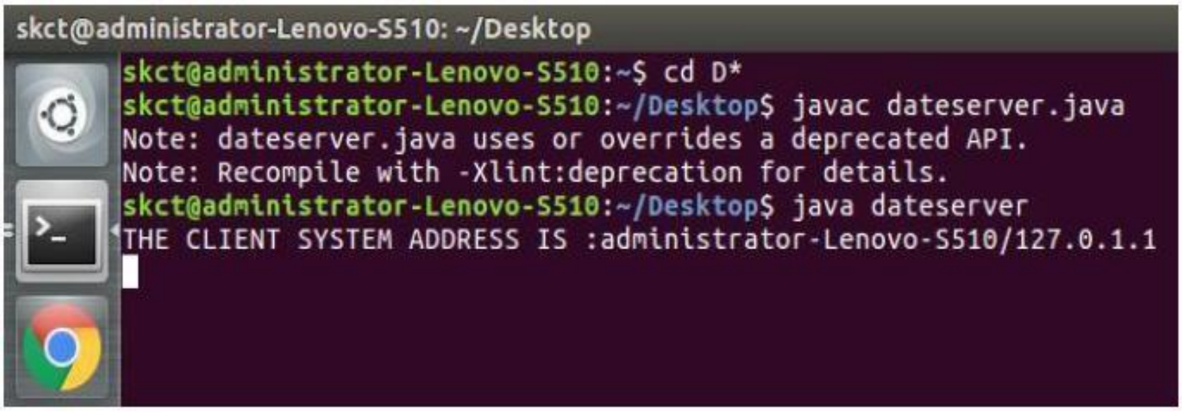
TCP Program:
dateserver.java

```
//import java packages
import java.net.*; import java.io.*; importjava.util.*;
/*… Register service on port 8020…*/
ss=new ServerSocket(8020);
/*… Wait and accept a connection…*/
s=ss.accept();
/*… Get a communication stream associated with the socket…*/
ps=new PrintStream(s.getOutputStream());
/* …To get system time…*/
Date d=new Date();
ps.println(d);
dis=new DataInputStream(s.getInputStream());
    inet=dis.readLine();System.out.println("THE CLIENT SYSTEM ADDRESS IS :"+inet);
/* …This method is used to request for closing or terminating an object…*/
ps.close();}}
```

dateclient.java

```
/* …Socket class is having a constructor through this Client program can request to server to get
connection…*/
Socket soc;
DataInputStream dis;
String sdate;
PrintStreamps;
/*…getLocalHost() method: Returns the name of the local computer…*/
InetAddressia=InetAddress.getLocalHost();
/*… Open your connection to a server, at port 8020…*/
soc=new Socket(ia,8020);
/*… Get an input file handle from the socket and read the input…*/
/*…getInputStream()-This method take the permission to write the data from client program to
server program and server program to client program…*/
dis=new DataInputStream(soc.getInputStream());
sdate=dis.readLine();
System.out.println("THE date in the server is:"+sdate);
/* …getOutputStream()-This method is used to take the permission to read data from client
system by the server or from the server system by the client…*/
ps=new PrintStream(soc.getOutputStream());
ps.println(ia);}
```

Output:



UDP Program:

Server.java

/*…import java packages…*/

import java.net.*; import java.io.*; importjava.util.*;

/*..receiving the packet from client…*/

DatagramPacketrp=new DatagramPacket(rd,rd.length); ss.receive(rp);

InetAddress ip= rp.getAddress(); int port=rp.getPort();

/*… getting system time…*/

Date d=new Date();

/*… converting it to String…*/

String time= d + "";

/*… converting that String to byte…*/

sd=time.getBytes();

/*…sending the data to the client…*/

DatagramPacketsp=new DatagramPacket(sd,sd.length,ip,port);

ss.send(sp);

Clientnew.java

/*…send the data to the server(data,length,ip address and port number)…*/

DatagramPacketsp=new DatagramPacket(sd,sd.length,ip,1234);

DatagramPacketrp=new DatagramPacket(rd,rd.length);

/*…To Send the data…*/

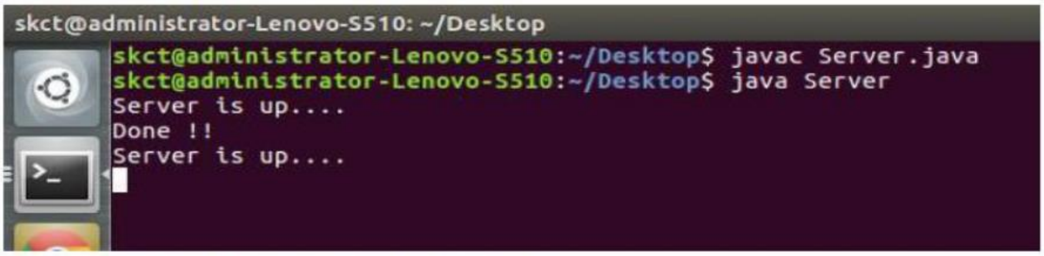
cs.send(sp); cs.receive(rp);

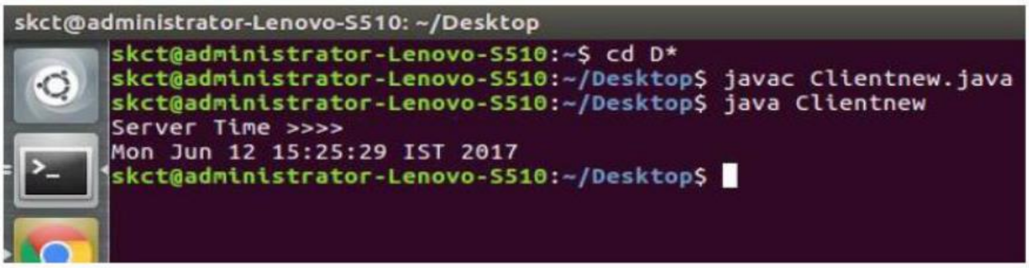
String time=new String(rp.getData()); System.out.println(time);

/*…This method is used to request for closing or terminating an object…*/

cs.close(); } }

Output:





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

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

(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 Clientrarp12
{
public static void main(String args[]){
try{
DatagramSocket client=new DatagramSocket();
InetAddress addr=InetAddress.getByName("127.0.0.1");
byte[] sendbyte=new byte[1024];
byte[] receivebyte=new byte[1024];
BufferedReader in=new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter the Physical address (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 Serverrarp12
{
public static void main(String args[]){
try{
DatagramSocket server=new DatagramSocket(1309);
while(true){
byte[] sendbyte=new byte[1024];
byte[] receivebyte=new byte[1024];
DatagramPacket receiver=new
DatagramPacket(receivebyte,receivebyte.length);
server.receive(receiver);
String str=new String(receiver.getData());
String s=str.trim();
//System.out.println(s);
InetAddress addr=receiver.getAddress();
int port=receiver.getPort();
String ip[]={"165.165.80.80","165.165.79.1"};
String mac[]={"6A:08:AA:C2","8A:BC:E3:FA"};
for(int i=0;i<ip.length;i++){
if(s.equals(mac[i]))
{
sendbyte=ip[i].getBytes();
DatagramPacket sender=new
DatagramPacket(sendbyte,sendbyte.length,addr,port);
server.send(sender);
break;
}}
break;
}}
catch(Exception e)
{
System.out.println(e);}
}
}
```

Output:

```
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
```

EXPERIMENT 4:
WRITE A CODE SIMULATING “PING” AND “TRACEROUTE” COMMANDS.

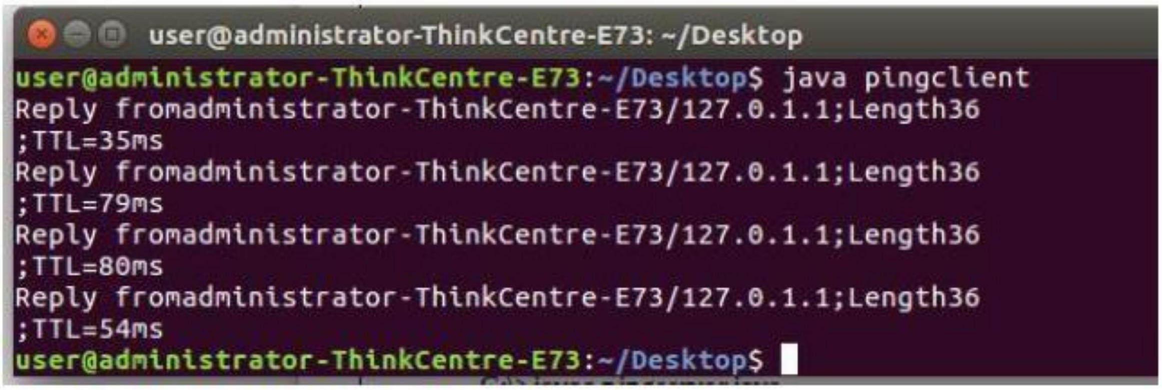
OBJECTIVE:
To Write The java program for simulating ping and traceroute commands.

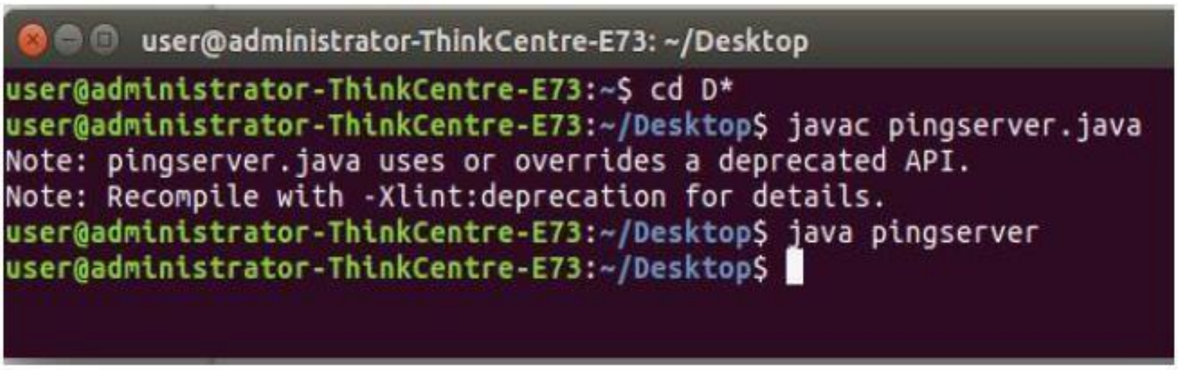
Program:

```
//pingclient.java
import   java.io.*;
import java.net.*;
import java.util.Calendar;
class pingclient
{
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();
}
}
//pingserver.java
import java.io.*;
import java.net.*;
import java.util.*;
import java.text.*;
class pingserver
{
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();
}
}
```

Output :





EXPERIMENT 5:

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.

Program :

//CLIENT CLASS

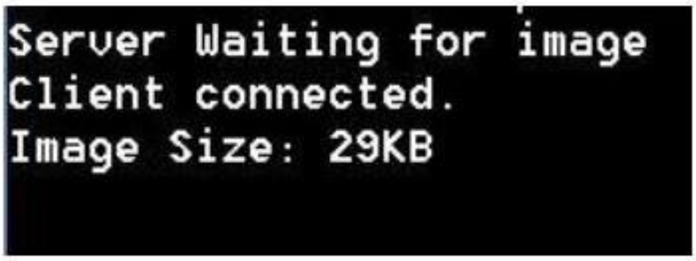
```
import javax.swing.*;
import  java.net.*;
import java.awt.image.*;
import javax.imageio.*;
import java.io.*;
import java.awt.image.BufferedImage;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
public class Client{
public static void main(String args[]) throws Exception{
Socket soc;
BufferedImage img = null;
soc=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.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 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(tru
e);
}
}
```

Output

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



EXPERIMENT 6:

WRITE A PROGRAM TO IMPLEMENT RPC (REMOTE PROCEDURE CALL)

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

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

Result :
Thus the program was implementing to implement RPC (remote procedure call)

RESULT:

Thus the Java-Program to implement Client - Server Communication using RPC was executed and output verified using various samples.

EXPERIMENT 7:
IMPLEMENTATION OF SUBNETTING.

OBJECTIVE:

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

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("\\."); //SPlit the string after every .
String split_bip[] = new String[4]; //split binary ip
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();

//Calculation of mask
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);

//Calculation of first address and last address
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; //convert cahracter 0,1 to integer 0,1
for(int i=31;i>31-bits;i–)//Get last address by ORing last n bits with 1
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
```

EXPERIMENT 8:

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

a. **Echo client and echo server**

OBJECTIVE

To write a java program for applications using TCP Sockets Links

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 :
Refer to Experiment 17.

b. <u>Chat</u>

```
//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=new DataInputStream(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:
Refer to Experiment 17.

c. **File Transfer**

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
jhfcgsauf
jbsdava
jbvuesagv
client end:
Enter the file name:sample.txt
Server response:
Computer networks
jhfcgsauf
jbsdava
jbvuesagv
client end:
Enter the new file name: net.txt
Computer networks
jhfcgsauf
jbsdava
jbvuesagv
Destination file
Computer networks
jhfcgsauf
jbsdava
jbvuesagv

EXPERIMENT 9:

APPLICATIONS USING TCP AND UDP SOCKETS LIKE ,

A. DNS

B. SNMP

C. FILE TRANSFER

a. DNS

OBJECTIVE

To write a java program for DNS application 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

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 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.listen();
}
/**
* 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 pdu = new PDU();
for (OID 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.version2c);
return target;
}}
```

OUT PUT

Hardware: x86 Family 6 Model 23 Stepping 10 AT/AT COMPATIBLE – Software:

Windows
2000 Version 5.1 (Build 2600 Multiprocessor Free)

b. File Transfer

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

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
jhfcgsauf
jbsdava
jbvuesagv
client
Enter the file name:
sample.txt
server
Computer networks
jhfcgsauf
jbsdava
jbvuesagv
client
Enter the new file name:
net.txt
Computer networks
jhfcgsauf
jbsdava
jbvuesagv
Destination file
Computer networks
jhfcgsauf
jbsdava
jbvuesagv

EXPERIMENT 10:
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 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 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.

EXPERIMENT 11:

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**

OBJECTIVE:
To study the link state routing flooding and distance vector routing.

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 narrower 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 floodingavailable, Uncontrolled Flooding and Controlled Flooding. Uncontrolled Flooding is the fatal law of flooding. All nodes have neighbors and route packets indefinitely. More than twoneighbors 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 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).

EXPERIMENT 12:

To learn handling and configuration of networking hardware like RJ-45 connector, CAT-6 cable, crimping tool, etc.

RJ45 Connector

RJ45 is a type of connector commonly used for Ethernet networking. It looks similar to a telephone jack, but is slightly wider. The "RJ" in RJ45 stands for "registered jack," since it is a standardized networking interface. The "45" simply refers to the number of the interface standard. Each RJ45 connector has eight pins, which means an RJ45 cable contains eight separate wires. Four of them are solid colors, while the other four are striped.

RJ45 cables can be wired in two different ways. One version is called T-568A and the other is T-568B. These wiring standards are listed below:

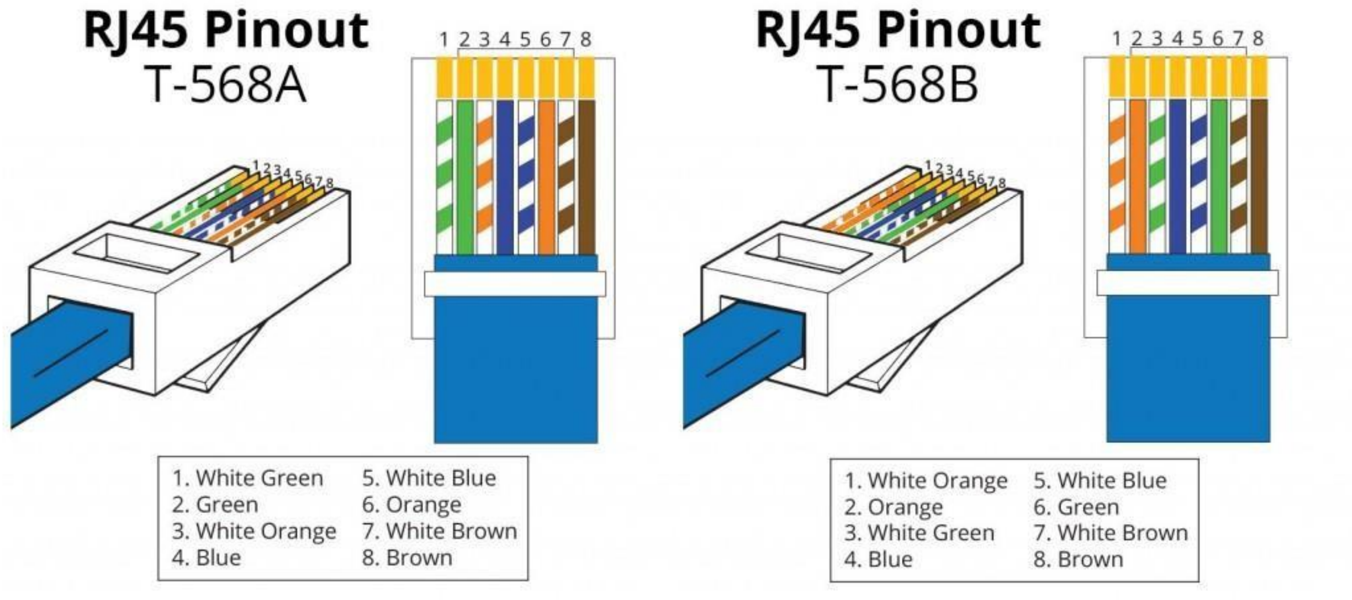
T-568A

1. White/Green (Receive +)
2. Green (Receive -)
3. White/Orange (Transmit +)
4. Blue
5. White/Blue
6. Orange (Transmit -)
7. White/Brown
8. Brown

T-568B

1. White/Orange (Transmit +)
2. Orange (Transmit -)
3. White/Green (Receive +)
4. Blue
5. White/Blue
6. Green (Receive -)
7. White/Brown
8. Brown

The T-568B wiring scheme is by far the most common, though many devices support the T-568A wiring scheme as well. Some networking applications require a crossover Ethernet cable, which has a T-568A connector on one end and a T-568B connector on the other. This type of cable is typically used for direct computer-to-computer connections when there is no router, hub, or switch available.



Cat 6 Cable

Category 6 is an [Ethernet cable] standard defined by the Electronic Industries Association (EIA) and Telecommunications Industry Association (TIA). Cat 6 is the sixth generation of twisted pair Ethernet cabling that is used in home and business networks. Cat 6 cabling is backward compatible with the Cat 5 and Cat 5e standards that preceded it.. Compared with Cat 5 and Cat 5e, Cat 6 features more stringent specifications for [crosstalk] and system noise. The cable standard also specifies performance of up to 250 MHz compared to 100 MHz for Cat 5 and Cat 5e. Cat 6 cable can be identified by the printing on the side of the cable sheath.

Working

Category 6 cables support Gigabit Ethernet data rates of 1 gigabit per second. They can accommodate 10 Gigabit Ethernet connections over a limited distance 164 feet for a single cable. Cat 6 cable contains four pairs of copper wire and uses all the pairs for signaling in order to obtain its high level of performance.

Other basic facts about Cat 6 cables include:

- The ends of a Cat 6 cable use the same RJ-45 standard connector as previous generations of Ethernet cables.
- The cable is identified as Cat 6 by printed text along the insulation sheath.
- An enhanced version of Cat 6 called Cat 6a supports up to 10 Gbps speeds

Limitations of Cat 6

- As with all other types of twisted pair EIA/TIA cabling, individual Cat 6 cable runs are limited to a maximum recommended length of 328 feet for their nominal connection speeds. As mentioned previously, Cat 6 cabling supports 10 Gigabit Ethernet connections, but not at this full distance.

Crimping tool

A crimping tool is a device used to conjoin two pieces of metal by deforming one or both of them in a way that causes them to hold each other. The result of the tool's work is called a crimp. A good example of crimping is the process of affixing a connector to the end of a cable. For instance, network cables and phone cables are created using a crimping tool (shown below) to join the RJ-45 and RJ-11 connectors to the both ends of either phone or Cat 5 cable.

Working

To use this crimping tool, each wire is first placed into the connector. Once all the wires are in the jack, the connectors with wires are placed into the crimping tool, and the handles are squeezed together. Crimping punctures the plastic connector and holds each of the wires, allowing for data to be transmitted through the connector.

EXPERIMENT 13:
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.

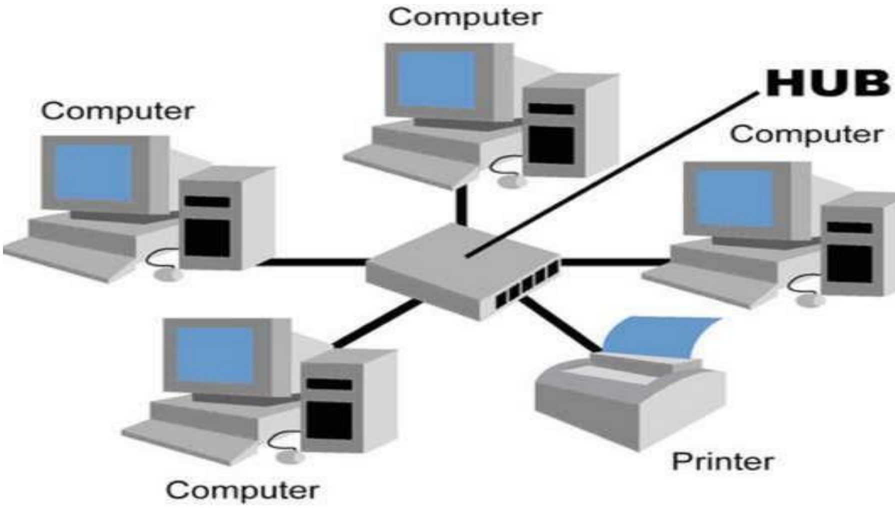
A router has two types of network element components organized onto separate planes:

- 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 examples 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 block.The intermediate device stores the message for a time being, after inspects it for errors, intermediate 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.

EXPERIMENT 14:
RUNNING AND USING SERVICES/COMMANDS LIKE PING, TRACE ROUTE, NSLOOKUP, ARP, TELNET, FTP, ETC.

Running and using Service Commands

1. **Ping**

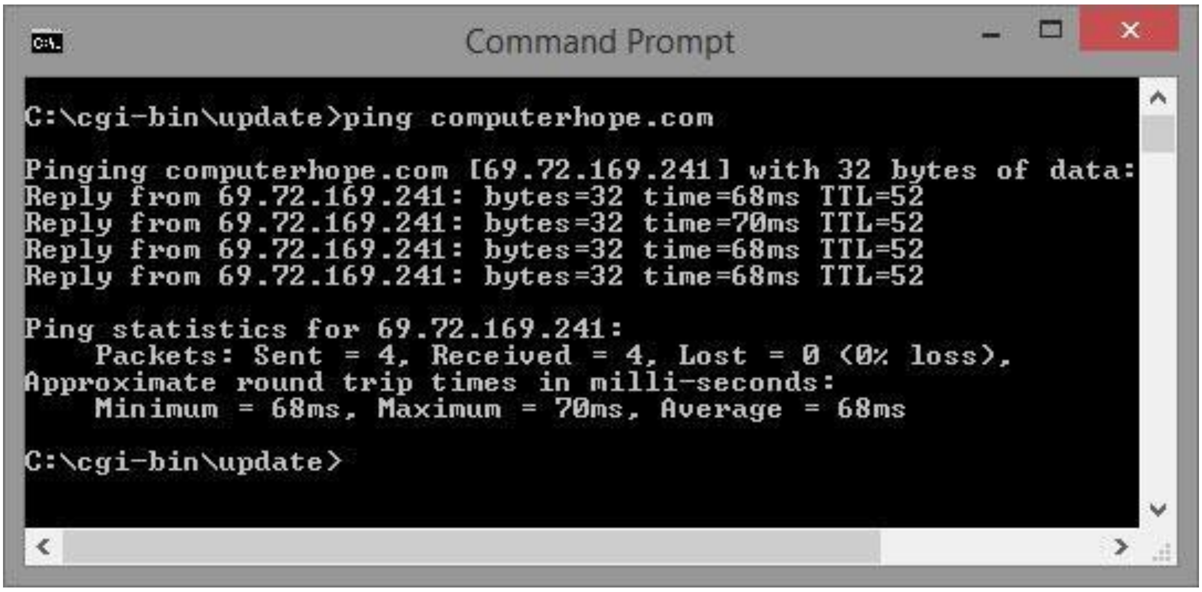
Ping is a basic Internet program that allows a user to verify that a particular IP address exists and can accept requests.
Ping is used diagnostically to ensure that a host computer the user is trying to reach is actually operating. Ping works by sending an Internet Control Message Protocol (ICMP) Echo Request to a specified interface on the network and waiting for a reply. Ping can be used for troubleshooting to test connectivity and determine response time.

Ping Command Syntax:
ping [-t] [-a] [-n count] [-l size] [-f] [-i TTL] [-v TOS] [-r count] [-s count] [-w timeout] [-R] [-S srcaddr] [-p] [-4] [-6] target [/?]

Example :

ping computerhope.com

ping 192.168.2.1

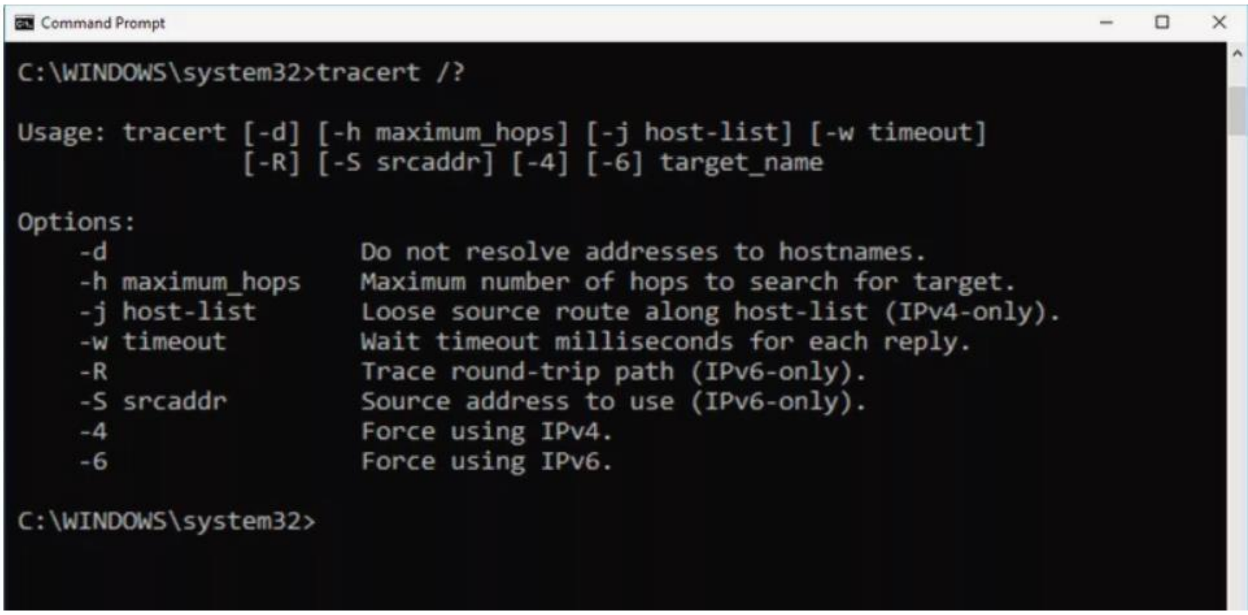


2. **Traceroute**

A traceroute is a function which traces the path from one network to another. It allows us to diagnose the source of many problems. The tracert command is a Command Prompt command that's used to show several details about the path that a packet takes from the computer or device you're on to whatever destination you specify.

Tracert command syntax:tracert [-d] [-h MaxHops] [-w TimeOut] [-4] [-6] target [/?]

Example:
tracert 192.168.1.1
tracert www.google.com



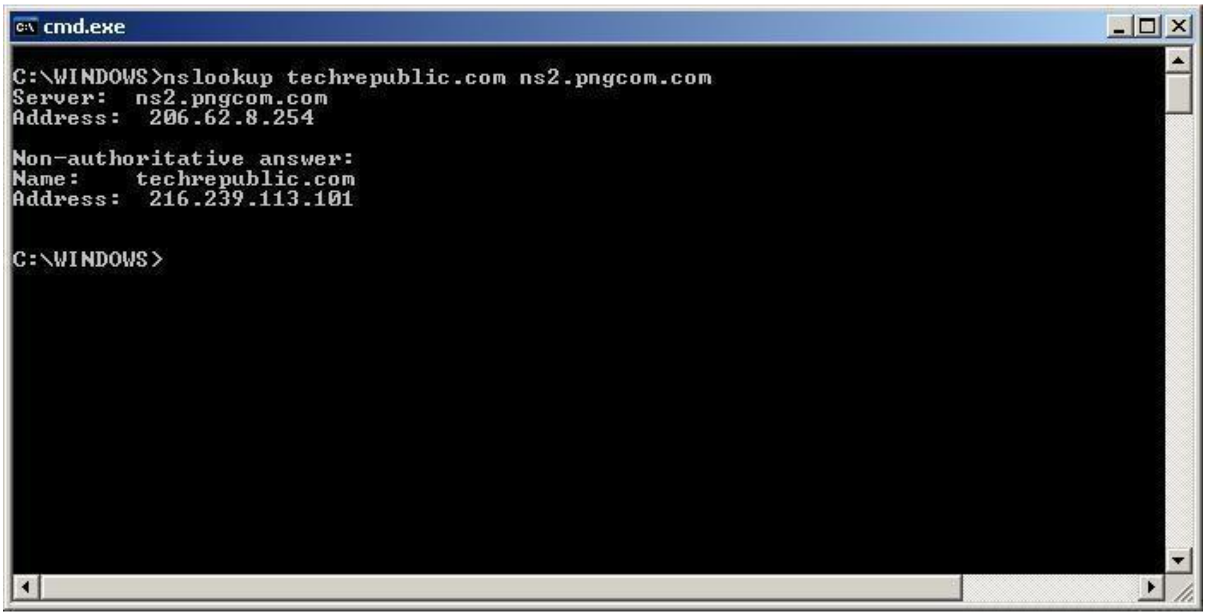
3. **Command nslookup**

The nslookup (which stands for name server lookup) command is a network utility program used to obtain information about internet servers. It finds name server information for domains by querying the Domain Name System.

Command nslookup sends a domain name query packet to a designated (or defaulted) domain name system (DNS) server. Depending on the system you are using, the default may be the local DNS name server at your service provider, some intermediate name server, or the root server system for the entire domain name system hierarchy.

<u>nslookup command syntax:</u>

nslookup [-*SubCommand* ...] [{*ComputerToFind*| [-*Server*]}]



4. **ARP(Address Resolution Protocol)**

ARP is used with the IP for mapping a 32-bit Internet Protocol address to a MAC address that is recognized in the local network specified in RFC 826. Once recognized, the server or networking device returns a response containing the required address.

ARP command syntax:

ARP -s inet_addr eth_adr [if_addr]

ARP -d inet_addr [if_addr]

ARP -a [inet_addr] [-N if_addr]

Example:

arp -a

arp -s 220.0.0.161 00-50-04-62-F7-23

```
C:\>arp -?

Displays and modifies the IP-to-Physical address translation tables used by
address resolution protocol (ARP).

ARP -s inet_addr eth_addr [if_addr]
ARP -d inet_addr [if_addr]
ARP -a [inet_addr] [-N if_addr]

  -a            Displays current ARP entries by interrogating the current
                protocol data.  If inet_addr is specified, the IP and Physical
                addresses for only the specified computer are displayed.  If
                more than one network interface uses ARP, entries for each ARP
                table are displayed.
  -g            Same as -a.
  inet_addr     Specifies an internet address.
  -N if_addr    Displays the ARP entries for the network interface specified
                by if_addr.
  -d            Deletes the host specified by inet_addr. inet_addr may be
                wildcarded with * to delete all hosts.
  -s            Adds the host and associates the Internet address inet_addr
                with the Physical address eth_addr.  The Physical address is
                given as 6 hexadecimal bytes separated by hyphens. The entry
                is permanent.
  eth_addr      Specifies a physical address.
  if_addr       If present, this specifies the Internet address of the
                interface whose address translation table should be modified.
                If not present, the first applicable interface will be used.
Example:
  > arp -s 157.55.85.212   00-aa-00-62-c6-09  .... Adds a static entry.
  > arp -a                                    .... Displays the arp table.

C:\>
```

5. **TelNet**
   Telnet is a user command and an underlying TCP/IP protocol for accessing remote computers. Through Telnet, an administrator or another user can access someone else's computer remotely. On the Web, HTTP and FTP protocols allow you to request specific files from remote computers, but not to actually be logged on as a user of that computer. With Telnet, you log on as a regular user with whatever privileges you may have been granted to the specific application and data on that computer.

   Telnet command syntax:
   telnet [/a] [/e <EscapeChar>] [/f <FileName>] [/l <UserName>] [/t {vt100 | vt52 | ansi | vtnt}] [<Host> [<Port>]] [/?]

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

H:\>Telnet 192.168.1.1 /?

telnet [-a][-e escape char][-f log file][-l user][-t term][host [port]]
 -a      Attempt automatic logon. Same as -l option except uses
         the currently logged on user's name.
 -e      Escape character to enter telnet client prompt.
 -f      File name for client side logging
 -l      Specifies the user name to log in with on the remote system.
         Requires that the remote system support the TELNET ENVIRON option.
 -t      Specifies terminal type.
         Supported term types are vt100, vt52, ansi and vtnt only.
 host    Specifies the hostname or IP address of the remote computer
         to connect to.
 port    Specifies a port number or service name.

H:\>telnet 192.168.1.1 -f C:\ROUTER_CFG.TXT
```

6.FTP(File Transfer Protocol)

File Transfer Protocol (FTP) is the commonly used protocol for exchanging files over the Internet. FTP uses the Internet's TCP/IP protocols to enable data transfer. FTP uses a client-server architecture, often secured with SSL/TLS. FTP promotes sharing of files via remote computers with reliable and efficient data transfer.

FTP command syntax:

FTP [-options] [-s:filename] [-w:buffer] [host]

Connect using FTP:

To connect to another computer using FTP at the MS-DOS prompt, command line, or Linux shell type FTP and press Enter. :

open ftp.example.com

Commands to run at the FTP: prompt

1.append local-file [remote-file] : Append a local file to a file on the remote computer.

2.ascii: Set the file transfer type to ASCII, the default. In ASCII text mode, character-set and end-of-line characters are converted as necessary.

3.bell: Toggle a bell to ring after each command. By default, the bell is off.

4.binary: Set the file transfer type to binary. Use `Binary' for transferring executable program files or binary data files e.g. Oracle

5.bye : End the FTP session and exit ftp

6.cd: Change the working directory on the remote host.

7.close : End the FTP session and return to the cmd prompt.

8.debug : Toggle debugging. When debug is on, FTP will display every command.

9.delete remote-file: Delete file on remote host.

10. dir [remote-directory] [local-file]: List a remote directory's files and subdirectories.(or save the listing to local-file)

11. disconnect: Disconnect from the remote host, retaining the ftp prompt.

12.get remote-file [local-file]: Copy a remote file to the local PC.

13. glob : Toggle the use of wildcard characters in local pathnames.By default, globbing is on.

14. hash : Toggle printing a hash (#) for each 2K data block transferred. By default, hash mark printing is off.

15. help [command] Display help for ftp command.

16. lcd [directory] Change the working directory on the local PC. By default, the working directory is the directory in which ftp was started.

17. literal argument: Send arguments, as-is, to the remote FTP host.

18. ls [remote-directory] [local-file] List a remote directory's files and folders.

19.mdelete remote-files [...] Delete files on remote host.

20. mget remote-files [...] Copy multiple remote files to the local PC.

21. status Display the current status of FTP connections and toggles.

22. trace Toggles packet tracing; trace displays the route of each packet

23. verbose Toggle verbose mode. By default, verbose is on.

24. ! command Run command on the local PC.

EXPERIMENT 15:
NETWORK PACKET ANALYSIS USING TOOLS LIKE WIRESHARK, TCPDUMP, ETC.

Network Packet Analysis Tools

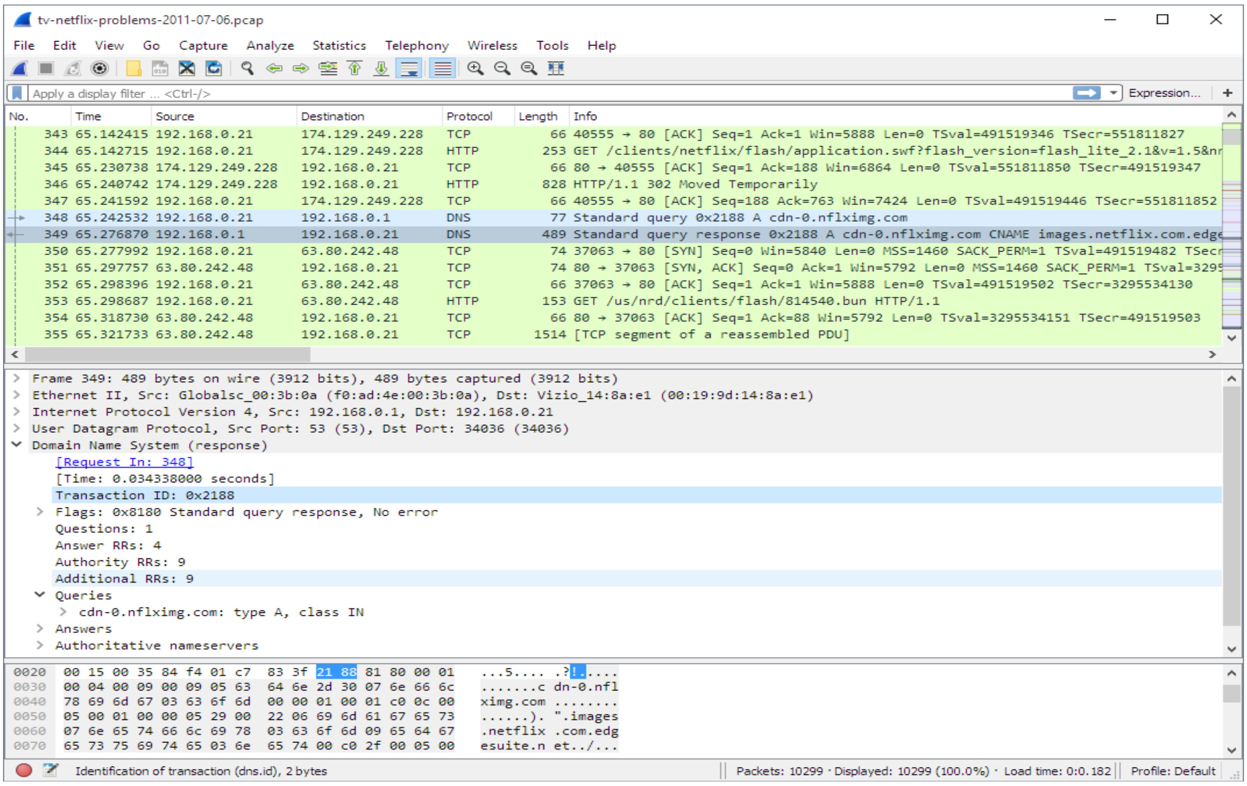
- A **packet analyzer** (also known as a **packet sniffer**) is a computer program or piece of computer hardware that can intercept and log traffic that passes over a digital network or part of a network.
- **Packet capture** is the process of intercepting and logging traffic.
- As data streams flow across the network, the sniffer captures each packet and, if needed, decodes the packet's raw data, showing the values of various fields in the packet, and analyzes its content according to the appropriate RFC or other specifications.
- A packet analyzer used for intercepting traffic on wireless networks is known as a **wireless analyzer** or **WiFi analyzer**.
- A packet analyzer can also be referred to as a network analyzer or protocol analyzerthough these terms also have other meanings.

Tools

1. Wireshark
2. Tcpdump

Wireshark

- **Wireshark** is a free and open-source packet analyzer.
- It is used for network troubleshooting, analysis, software and communications protocol development, and education. Originally named **Ethereal**, the project was renamed Wireshark in May 2006 due to trademark issues.
- Wireshark is cross-platform, using the Qt widget toolkit in current releases to implement its user interface, and using pcap to capture packets.
- It runs on Linux, macOS, BSD, Solaris, some other Unix-like operating systems, and Microsoft Windows.
- There is also a terminal-based (non-GUI) version called TShark.
- Wireshark, and the other programs distributed with it such as TShark, are free software, released under the terms of the GNU General Public License.



Functionality

- Wireshark is very similar to tcpdump, but has a graphical front-end, plus some integrated sorting and filtering options.
- Wireshark lets the user put network interface controllers into promiscuous mode (if supported by the network interface controller), so they can see all the traffic visible on that interface including unicast traffic not sent to that network interface controller's MAC address.
- However, when capturing with a packet analyzer in promiscuous mode on a port on a network switch, not all traffic through the switch is necessarily sent to the port where the capture is done, so capturing in promiscuous mode is not necessarily sufficient to see all network traffic.
- Port mirroring or various network taps extend capture to any point on the network. Simple passive taps are extremely resistant to tampering.
- On GNU/Linux, BSD, and macOS, with libpcap 1.0.0 or later, Wireshark 1.4 and later can also put wireless network interface controllers into monitor mode.
- If a remote machine captures packets and sends the captured packets to a machine running Wireshark using the TZSP protocol or the protocol used by OmniPeek, Wireshark dissects those packets, so it can analyze packets captured on a remote machine at the time that they are captured.

Tcpdump

- **Tcpdump** is a common packet analyzer that runs under the command line.
- It allows the user to display TCP/IP and other packets being transmitted or received over a network to which the computer is attached.
- Distributed under the BSD license, tcpdump is free software.
- Tcpdump works on most Unix-like operating systems: Linux, Solaris, FreeBSD, DragonFly BSD, NetBSD, OpenBSD, OpenWrt, macOS, HP-UX 11i, and AIX.
- In those systems, tcpdump uses the libpcap library to capture packets.
- The port of tcpdump for Windows is called WinDump; it uses WinPcap, the Windows port of libpcap.

Priveleges Required

- In some Unix-like operating systems, a user must have super user privileges to use tcpdump because the packet capturing mechanisms on those systems require elevated privileges. However, the -Z option may be used to drop privileges to a specific unprivileged user after capturing has been set up.
- In other Unix-like operating systems, the packet capturing mechanism can be configured to allow non-privileged users to use it; if that is done, super user privileges are not required.

EXPERIMENT 16:

NETWORK SIMULATION USING TOOLS LIKE CISCO PACKET TRACER, NETSIM, OMNET++, NS2, NS3, ETC.

Network simulation tools

Network Simulator provides an integrated, versatile, easy-to-use GUI-based network designer tool to design and simulate a network with SNMP, TL1, TFTP, FTP, Telnet and Cisco IOS device.

List of Network Simulators:

There are different network simulators which offer different features. we have listed different network simulators and sample program code

- Ns2 (Network Simulator 2).
- Ns3 (Network Simulator 3).
- OPNET.
- OMNeT++.
- NetSim.
- REAL.
- QualNet.
- J-Sim.

Network simulator

A network simulator is software that predicts the behavior of a computer network. Since communication networks have become too complex for traditional analytical methods to provide an accurate understanding of system behavior, network simulators are used. In simulators, the computer network is modeled with devices, links, applications etc. and the network performance is reported. Simulators come with support for the most popular technologies and networks in use today such as Wireless LANs, mobile ad hoc networks, wireless sensor networks, vehicular ad hoc networks, cognitive radio networks, LTE / LTE- 5G, Internet of Things (IoT) etc.

Simulations

Most of the commercial simulators are GUI driven, while some network simulators are CLI driven. The network model / configuration describes the network (nodes, routers, switches, links) and the events (data transmissions, packet error etc.). Output results would include network level metrics, link metrics, device metrics etc. Further, drill down in terms of simulations trace files would also be available. Trace files log every packet, every event that occurred in the simulation and are used for analysis. 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.*

Network emulation

Network emulation allows users to introduce real devices and applications into a test network (simulated) that alters packet flow in such a way as to mimic the behavior of a live network. Live traffic can pass through the simulator and be affected by objects within the simulation.

The typical methodology is that real packets from a live application are sent to the emulation server (where the virtual network is simulated). The real packet gets 'modulated' into a simulation packet. The simulation packet gets demodulated into a real packet after experiencing effects of loss, errors, delay, jitter etc., thereby transferring these network effects into the real packet. Thus it is as-if the real packet flowed through a real network but in reality it flowed through the simulated network.

Emulation is widely used in the design stage for validating communication networks prior to deployment.

Uses of network simulators

Network simulators provide a cost-effective method for

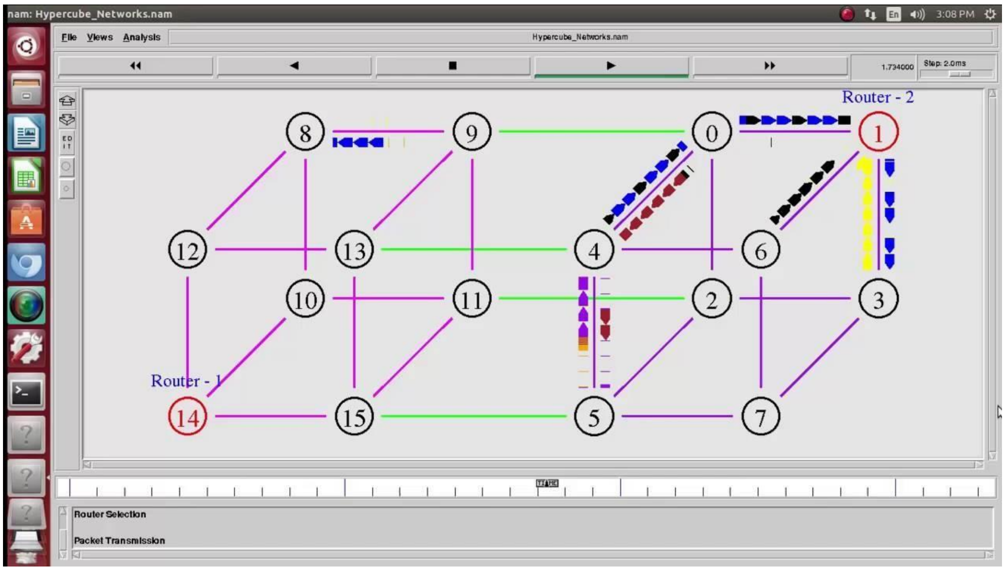
- Network design validation for enterprises / data centers / sensor networks etc.
- Network R & D (More than 70% of all Network Research paper reference a network simulator)[*citation needed*]
- Defense applications such as HF / UHF / VHF Radio based MANET Radios, Naval communications, Tactical data links etc.
- LTE, LTE-Adv, IOT, VANET simulations
- Education - Lab experimentation and R & D. Most universities use a network simulator for teaching / R & D since its too expensive to buy hardware equipment

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

- Model the network topology specifying the nodes on the network and the links between those nodes
- Model the application flow (traffic) between the nodes
- Providing network performance metrics as output
- Visualization of the packet flow
- Technology / protocol evaluation and device designs
- Logging of packet/events for drill down analyses / debugging

NS2 (Network Simulator 2)

- *It is a discrete event simulator that provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless networks.*
- *It use C++ and OTcl languages.*
- *Sample ns2 code. In which there are four different nodes are available and two different routers.*



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

Ns began as a variant of the REAL network simulator in 1989 and has evolved substantially over the past few years. In 1995 ns development was supported by DARPA through the VINT project at LBL, Xerox PARC, UCB, and USC/ISI. Currently ns development is supported through DARPA with SAMAN and through NSF with CONSER, both in collaboration with other researchers including ACIRI. Ns has always included substantal contributions from other researchers, including wireless code from the UCB Daedelus and CMU Monarch projects and Sun Microsystems.

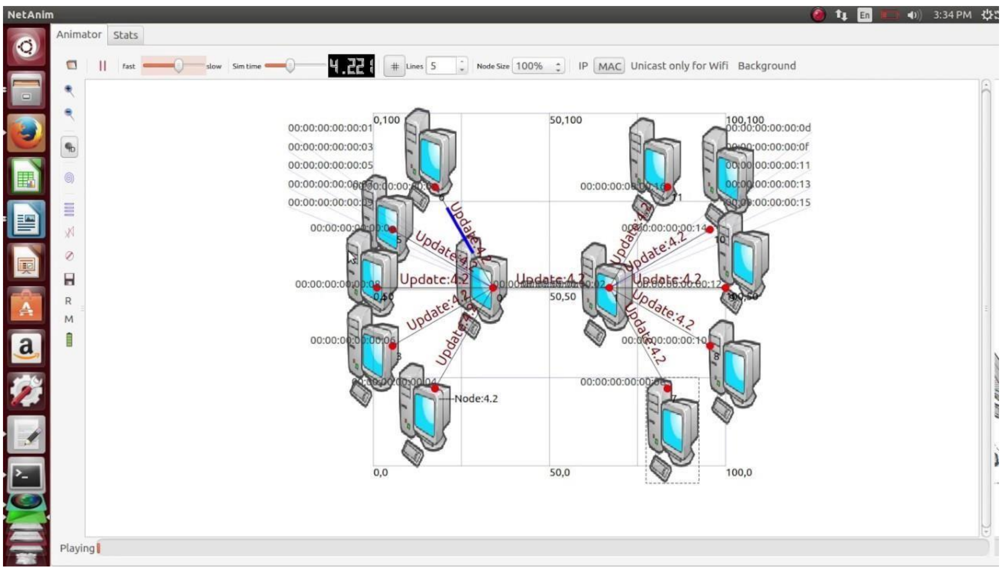
NS3 (Network Simulator 3):

In 2006, a team led by Tom Henderson, George Riley, Sally Floyd, and Sumit Roy, applied for and received funding from the U.S. National Science Foundation (NSF) to build a replacement for ns-2, called ns-3. This team collaborated with the Planete project of INRIA at Sophia Antipolis, with Mathieu Lacage as the software lead, and formed a new open source project.

In the process of developing ns-3, it was decided to completely abandon backward-compatibility with ns-2. The new simulator would be written from scratch, using the C++programming language. Development of ns-3 began in July 2006.

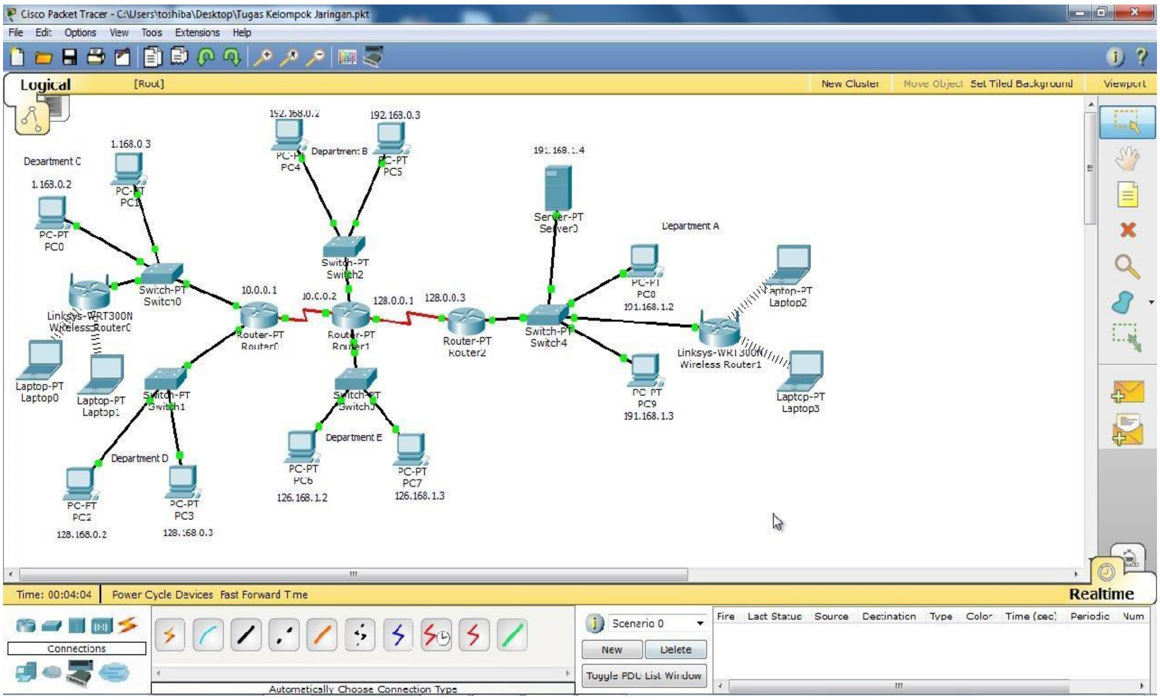
Current status of the three versions is:

- Ns3development stopped when ns-2 was founded. It is no longer developed nor maintained.
- Ns3development stopped around 2010. It is no longer developed nor maintained.
- Ns3is actively being developed and maintained.
- Ns3 uses C++ and Python languages for simulating the script.
- C++ used for implementation of simulation and core model. Ns-3 is built as a library which may be statically or dynamically linked to a C++ main program.
- Python: C++ wrapped by Python. Python programs to import an "ns3" module
- Sample code for ns3.



Cisco packet tracer

Packet Tracer is a cross-platform visual simulation tool designed by Cisco Systems that allows users to create network topologies and imitate modern computer networks. The software allows users to simulate the configuration of Cisco routers and switches using a simulated command line interface. Packet Tracer makes use of a drag and drop user interface, allowing users to add and remove simulated network devices as they see fit. The software is mainly focused towards Certified Cisco Network Associate Academy students as an educational tool for helping them learn fundamental CCNA concepts. Previously students enrolled in a CCNA Academy program could freely download and use the tool free of charge for educational use.



Overview

Packet Tracer can be run on Linux and Microsoft Windows and also macOS. Similar Android and iOS apps are also available. Packet Tracer allows users to create simulated network topologies by dragging and dropping routers, switches and various other types of network devices. A physical connection between devices is represented by a "cable" item. Packet Tracer supports an array of simulated Application Layer protocols, as well as basic routing with RIP, OSPF, EIGRP, BGP, to the extents required by the current CCNA curriculum. As of version 5.3, Packet Tracer also supports the Border Gateway Protocol.

In addition to simulating certain aspects of computer networks, Packet Tracer can also be used for collaboration. As of Packet Tracer 5.0, Packet Tracer supports a multi-user system that enables multiple users to connect multiple topologies together over a computer network.[6] Packet Tracer also allows instructors to create activities that students have to complete.[2] Packet Tracer is often used in educational settings as a learning aid. Cisco Systems claims that Packet Tracer is useful for network experimentation.

Role in Education

Packet Tracer allows students to design complex and large networks, which is often not feasible with physical hardware, due to costs. Packet Tracer is commonly used by CCNA Academy students, since it is available to them for free. However, due to functional limitations, it is intended by CISCO to be used only as a learning aid, not a replacement for Cisco routers and switches. The application itself only has a small number of features found within the actual hardware running a current Cisco IOS version. Thus, Packet Tracer is unsuitable for modelling production networks. It has a limited command set, meaning it is not possible to practice all of the IOS commands that might be required. Packet Tracer can be useful for understanding abstract networking concepts, such as the Enhanced Interior Gateway Routing Protocol by animating these elements in a visual form. Packet Tracer is also useful in education by providing additional components, including an authoring system, network protocol simulation and improving knowledge an assessment system.

Netsim

1) It has an object-oriented system modeling and simulation (M&S) environment to support simulation and analysis of voice and data communication scenarios for High Frequency Global Communication Systems (HFGCS).

2) NetSim use java as a programming language it creates applet and linked into HTML document for viewable on the java-compatible browser.

Netsim Standard version

1. Easy to use GUI allows users to simply drag and drop devices, links and applications.
2. Results dashboard provides appealing simulation performance reports with tables & graphs.
3. Inbuilt graphing with extensive formatting (axes, colours, zoom, titles etc).
4. Wide range of technologies including the latest in IOT, WSN, MANET, Cognitive Radio, 802.11 n / ac, TCP, BIC / CUBIC, Rate adaptation with packet and event tracing.
5. Online debug capability and ability to "watch" all variables.
6. Run animation in parallel for immediate visual feedback.

EXPERIMENT 17:

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.

PROGRAM:

```
//TCP Date Server--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; Stringinet;
try
{ ss = new ServerSocket(4444); System.out.println("Press
Ctrl+C to quit"); while(true){cs = ss.accept();
ps = new PrintStream(cs.getOutputStream()); Date d = new Date();
ps.println(d);
dis = new BufferedReader(new InputStreamReader(cs.getInputStream())); inet
= dis.readLine();
System.out.println("Client System/IP address is :"+ inet); ps.close();
dis.close();
}
}
catch(IOException e)
{
System.out.println("The exception is :" + e);
}
}
}
```

```
// TCP Date Client--tcpdateclient.java
import java.net.*;
import java.io.*; class tcpdateclient
{
public static void main (String args[])
{

    Socket soc; BufferedReader dis; String sdate; PrintStream ps;
    try
    {
     InetAddress ia = InetAddress.getLocalHost(); if (args.length == 0)
     soc = new Socket(InetAddress.getLocalHost(),4444); else
     soc = new Socket(InetAddress.getByName(args[0]),4444);
     dis = new BufferedReader(newInputStreamReader(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(IOException e)
{
System.out.println("THE EXCEPTION is :" + e);
} } }
```

OUTPUT

Server:

```
$ javac
tcpdateserver.java $
java tcpdateserver
Press Ctrl+C to quit
Client System/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
The date/time on server is: Wed Jul 06 07:12:03 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.

OBJECTIVE: To implementation of echo client server using TCP/IP

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 =
new BufferedReader(new InputStreamReader(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 = new BufferedReader(new InputStreamReader(sock.getInputStream()));
fromUser = new BufferedReader(new InputStreamReader(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.

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 = new PrintWriter(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(new
        InputStreamReader(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.

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 = 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 toCleint : " );
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();

}
}
```

// 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(new InputStreamReader
(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();
}
}
```

OUTPUT

Server

```
$ 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
```

Client

```
$ 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.

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 IPADDRESS=%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
```

Result

Hence the program to develop a client that contacts a given DNS server to resolve a given host name is executed successfully.

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

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

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
```

RESULT:

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

Experiment Beyond Syllabus

EXPERIMENT 1:
PROGRAMMING USING RAW SOCKETS.

A raw socket is used to receive raw packets. This means packets received at the Ethernet layer will directly pass to the raw socket. Stating it precisely, a raw socket bypasses the normal TCP/IP processing and sends the packets to the specific user application (see Figure 1).

A raw socket vs other sockets
Other sockets like stream sockets and data gram sockets receive data from the transport layer that contains no headers but only the payload. This means that there is no information about the source IP address and MAC address. If applications running on the same machine or on different machines are communicating, then they are only exchanging data. The purpose of a raw socket is absolutely different. A raw socket allows an application to directly access lower level protocols, which means a raw socket receives un-extracted packets (see Figure 1). There is no need to provide the port and IP address to a raw socket, unlike in the case of stream and datagram sockets.

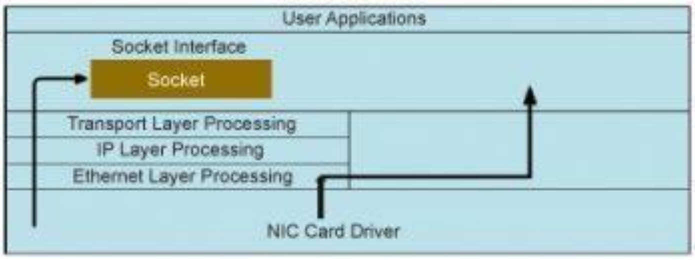


Figure 1: Graphical demonstration of how a raw socket works compared to other sockets

A packet sniffer with a raw socket: To develop a packet sniffer, you first have to open a raw socket. Only processes with an effective user ID of 0 or the *CAP_NET_RAW* capability are allowed to open raw sockets. So, during the execution of the program, you have to be the root user.

Opening a raw socket: To open a socket, you have to know three things – the socket family, socket type and protocol. For a raw socket, the socket family is *AF_PACKET*, the socket type is *SOCK_RAW*and for the protocol, see the if_ether.h header file. To receive all packets, the macro is *ETH_P_ALL* and to receive IP packets, the macro is *ETH_P_IP* for the protocol field.

```
int sock_r;
sock_r=socket(AF_PACKET,SOCK_RAW,htons(ETH_P_ALL));
if(sock_r<0)
{
printf(“error in socket\n”);
return -1;
}
```

Reception of the network packet: After successfully opening a raw socket, it’s time to receive network packets, for which you need to use the *recvfrom api.* We can also use the *recv api*. But recvfrom provides additional information.

```
unsigned char *buffer = (unsigned char *) malloc(65536); //to receive data
memset(buffer,0,65536);
struct sockaddr saddr;
int saddr_len = sizeof (saddr);

//Receive a network packet and copy in to buffer
buflen=recvfrom(sock_r,buffer,65536,0,&saddr,(socklen_t *)&saddr_len);
if(buflen<0)
{
printf("error in reading recvfrom function\n");
return -1;
}
```

Extracting the Ethernet header: Now that we have the network packets in our buffer, we will get information about the Ethernet header. The Ethernet header contains the physical address of the source and destination, or the MAC address and protocol of the receiving packet. The *if_ether.h* header contains the structure of the Ethernet header (see Figure 5). Now, we can easily access these fields:

```
struct ethhdr *eth = (struct ethhdr *)(buffer);
printf("\nEthernet Header\n");
printf("\t|-Source Address : %.2X-%.2X-%.2X-%.2X-%.2X-%.2X\n",eth->h_source[0],eth->h_source[1],
eth->h_source[2],eth->h_source[3],eth->h_source[4],eth->h_source[5]);
printf("\t|-Destination Address : %.2X-%.2X-%.2X-%.2X-%.2X-%.2X\n",eth->h_dest[0],
eth->h_dest[1],eth->h_dest[2],eth->h_dest[3],eth->h_dest[4],eth->h_dest[5]);
printf("\t|-Protocol : %d\n",eth->h_proto);
```

h_proto gives information about the next layer. If you get 0x800 *(ETH_P_IP)*, it means that the next header is the IP header. Later, we will consider the next header as the IP header.

Note 1: *The physical address is 6 bytes*.

Note 2: *We can also direct the output to a file for better understanding.*

```
fprintf(log_txt,"\t|-Source Address : %.2X-%.2X-%.2X-%.2X-%.2X-%.2X\n",eth->h_source[0],eth->h_source[1],
eth->h_source[2],eth->h_source[3],eth->h_source[4],eth->h_source[5]);
```

Use *fflush* to avoid the input-output buffer problem when writing into a file.

Extracting the IP header: The IP layer gives various pieces of information like the source and destination IP address, the transport layer protocol, etc. The structure of the IP header is defined in the *ip.h header file* (see Figure 6). Now, to get this information, you need to increment your buffer pointer by the size of the Ethernet header because the IP header comes after the Ethernet header:

```
unsigned short iphdrlen;
struct iphdr *ip = (struct iphdr*)(buffer + sizeof(struct ethhdr));
memset(&source, 0, sizeof(source));
source.sin_addr.s_addr = ip->saddr;
memset(&dest, 0, sizeof(dest));
dest.sin_addr.s_addr = ip->daddr;
fprintf(log_txt, "\t|-Version : %d\n",(unsigned int)ip->version);
fprintf(log_txt , "\t|-Internet Header Length : %d DWORDS or %d Bytes\n",
(unsigned int)ip->ihl,((unsigned int)(ip->ihl))*4);
fprintf(log_txt , "\t|-Type Of Service : %d\n",(unsigned int)ip->tos);

fprintf(log_txt , "\t|-Total Length : %d Bytes\n",ntohs(ip->tot_len));

fprintf(log_txt , "\t|-Identification : %d\n",ntohs(ip->id));

fprintf(log_txt , "\t|-Time To Live : %d\n",(unsigned int)ip->ttl);

fprintf(log_txt , "\t|-Protocol : %d\n",(unsigned int)ip->protocol);

fprintf(log_txt , "\t|-Header Checksum : %d\n",ntohs(ip->check));

fprintf(log_txt , "\t|-Source IP : %s\n", inet_ntoa(source.sin_addr));

fprintf(log_txt , "\t|-Destination IP : %s\n",inet_ntoa(dest.sin_addr));
```

The transport layer header

There are various transport layer protocols. Since the underlying header was the IP header, we have various IP or Internet protocols. You can see these protocols in the */etc/protocls file.* The TCP and UDP protocol structures are defined in tcp.h and udp.h respectively. These structures provide the port number of the source and destination. With the help of the port number, the system gives data to a particular application. The size of the IP header varies from 20 bytes to 60 bytes. We can calculate this from the IP header field or IHL. IHL means Internet Header Length (IHL), which is the number of 32-bit words in the header. So we have to multiply the IHL by 4 to get the size of the header in bytes:

```
struct iphdr *ip = (struct iphdr *)( buffer + sizeof(struct ethhdr) );
/* getting actual size of IP header*/
iphdrlen = ip->ihl*4;
/* getting pointer to udp header*/
struct tcphdr *udp=(struct udphdr*)(buffer + iphdrlen + sizeof(struct ethhdr));
```

We now have the pointer to the UDP header. So let’s check some of its fields.

```
fprintf(log_txt , "\t|-Source Port : %d\n" , ntohs(udp->source));
fprintf(log_txt , "\t|-Destination Port : %d\n" , ntohs(udp->dest));
fprintf(log_txt , "\t|-UDP Length : %d\n" , ntohs(udp->len));
fprintf(log_txt , "\t|-UDP Checksum : %d\n" , ntohs(udp->check));
```

Similarly, we can access the TCP header field.

Extracting data

After the transport layer header, there is data payload remaining. For this, we will move the pointer to the data, and then print.

```
unsigned char * data = (buffer + iphdrlen + sizeof(struct ethhdr) + sizeof(struct udphdr));
```

Now, let's print data, and for better representation, let us print 16 bytes in a line.

```
int remaining_data = buflen - (iphdrlen + sizeof(struct ethhdr) + sizeof(struct udphdr));
for(i=0;i<remaining_data;i++)
{
if(i!=0 && i%16==0)
fprintf(log_txt,"\n");
fprintf(log_txt," %.2X ",data[i]);
}
```

Sending packets with a raw socket

To send a packet, we first have to know the source and destination IP addresses as well as the MAC address. Use your friend's *MAC & IP* address as the destination IP and MAC address. There are two ways to find out your IP address and MAC address:

1. Enter *ifconfig* and get the IP and MAC for a particular interface.
2. Enter *ioctl* and get the IP and MAC. The second way is more efficient and will make your program machine-independent, which means you should not enter *ifconfig* in each machine.

Opening a raw socket

To open a raw socket, you have to know three fields of socket *API — Family- AF_PACKET*, *Type- SOCK_RAW* and for the protocol, let's use *IPPROTO_RAW* because we are trying to send an IP packet. *IPPROTO_RAW* macro is defined in the in.h header file:

```
sock_raw=socket(AF_PACKET,SOCK_RAW,IPPROTO_RAW);
if(sock_raw == -1)
printf("error in socket");
```

Getting the index of the interface to send a packet

There may be various interfaces in your machine like loopback, wired interface and wireless interface. So you have to decide the interface through which we can send our packet. After deciding on the interface, you have to get the index of that interface. For this, first give the name of the interface by setting the field.

Getting the IP address of the interface

For this, use the SIOCGIFADDR macro:

```
struct ifreq ifreq_ip;
memset(&ifreq_ip,0,sizeof(ifreq_ip));
strncpy(ifreq_ip.ifr_name,"wlan0",IFNAMSIZ-1);//giving name of Interface
if(ioctl(sock_raw,SIOCGIFADDR,&ifreq_ip)<0) //getting IP Address
{
printf("error in SIOCGIFADDR \n");
}
```

Constructing the Ethernet header After getting the index, as well as the MAC and IP addresses of an interface, it's time to construct the Ethernet header. First, take a buffer in which you will place all information like the Ethernet header, IP header, UDP header and data. That buffer will be your packet.

```
sendbuff=(unsigned char*)malloc(64); // increase in case of more data
memset(sendbuff,0,64);
```

To construct the Ethernet header, fill all the fields of the ethhdr structure:

```
struct ethhdr *eth = (struct ethhdr *)(sendbuff);

eth->h_source[0] = (unsigned char)(ifreq_c.ifr_hwaddr.sa_data[0]);
eth->h_source[1] = (unsigned char)(ifreq_c.ifr_hwaddr.sa_data[1]);
eth->h_source[2] = (unsigned char)(ifreq_c.ifr_hwaddr.sa_data[2]);
eth->h_source[3] = (unsigned char)(ifreq_c.ifr_hwaddr.sa_data[3]);
eth->h_source[4] = (unsigned char)(ifreq_c.ifr_hwaddr.sa_data[4]);
eth->h_source[5] = (unsigned char)(ifreq_c.ifr_hwaddr.sa_data[5]);

/* filling destination mac. DESTMAC0 to DESTMAC5 are macro having octets of mac address. */
eth->h_dest[0] = DESTMAC0;
eth->h_dest[1] = DESTMAC1;
eth->h_dest[2] = DESTMAC2;
eth->h_dest[3] = DESTMAC3;
eth->h_dest[4] = DESTMAC4;
eth->h_dest[5] = DESTMAC5;

eth->h_proto = htons(ETH_P_IP); //means next header will be IP header

/* end of ethernet header */
total_len+=sizeof(struct ethhdr);
```

Constructing the IP header: To construct the IP header, increment sendbuff by the size of the Ethernet header and fill each field of the *iphdr* structure. Data after the IP header is called the payload for the IP header and, in the same way, data after the Ethernet header is called the payload for the Ethernet header. In the IP header, there is a field called Total Length, which contains the size of the IP header plus the payload. To know the size of the payload of the IP header, you must know the size of the UDP header and the UDP payload. So, some field of the *iphdr structure* will get the value after filling the UDP header field.

```
struct iphdr *iph = (struct iphdr*)(sendbuff + sizeof(struct ethhdr));
iph->ihl = 5;
iph->version = 4;
iph->tos = 16;
iph->id = htons(10201);
iph->ttl = 64;
iph->protocol = 17;
iph->saddr = inet_addr(inet_ntoa((((struct sockaddr_in *)&(ifreq_ip.ifr_addr))->sin_addr)));
```

```
iph->daddr = inet_addr("destination_ip"); // put destination IP address

total_len += sizeof(struct iphdr);
```

Construct the UDP header: Constructing the UDP header is very similar to constructing the IP header. Assign values to the fields of the *udphdr* structure. For this, increment the *sendbuff* pointer by the size of the Ethernet and the IP headers.

```
struct udphdr *uh = (struct udphdr *)(sendbuff + sizeof(struct iphdr) + sizeof(struct ethhdr));

uh->source = htons(23451);
uh->dest = htons(23452);
uh->check = 0;

total_len+= sizeof(struct udphdr);
```

Like the IP header, the UDP also has the field len, which contains the size of the UDP header and its payload. So, first, you have to know the UDP payload, which is the actual data that will be sent.