UNIVERSITY COLLEGE OF ENGINEERING VILLUPURAM

DEPARTMENT OF INFORMATION TECHNOLOGY

CS8581

NETWORKS LABORATORY

Prepared by

Mr.P.TAMILARASU, TF/IT, UCEV

DATE: **1.NETWORKING COMMANDS**

AIM:

To study the basic networking commands.

NETWORKING COMMANDS:

C:\>arp –a: ARP is short form of address resolution protocol, It will show the IP address of your computer along with the IP address and MAC address of your router.

C:\>hostname: This is the simplest of all TCP/IP commands. It simply displays the name of your computer.

C:\>ipconfig: The ipconfig command displays information about the host (the computer your sitting at)computer TCP/IP configuration.

C:\>ipconfig /all: This command displays detailed configuration information about your TCP/IP connection including Router, Gateway, DNS, DHCP, and type of Ethernet adapter in your system.

C:\>Ipconfig /renew: Using this command will renew all your IP addresses that you are currently (leasing) borrowing from the DHCP server. This command is a quick problem solver if you are having connection issues, but does not work if you have been configured with a static IP address.

C:\>Ipconifg /release: This command allows you to drop the IP lease from the DHCP server.

C:\>ipconfig /flushdns: This command is only needed if you're having trouble with your networks DNS configuration. The best time to use this command is after network configuration frustration sets in, and you really need the computer to reply with flushed.

C:\>nbtstat –a: This command helps solve problems with NetBIOS name resolution. (Nbt stands for NetBIOS over TCP/IP)

C:\>netdiag: Netdiag is a network testing utility that performs a variety of network diagnostic tests, allowing you to pinpoint problems in your network. Netdiag isn't installed by default, but can be installed from the Windows XP CD after saying no to the install. Navigate to the CD ROM drive letter and open the support\tools folder on the XP CD and click the setup.exe icon in the support\tools folder.

C:\>netstat: Netstat displays a variety of statistics about a computers active TCP/IP connections. This tool is most useful when you're having trouble with TCP/IP applications such as HTTP, and FTP.

C:\>nslookup: Nslookup is used for diagnosing DNS problems. If you can access a resource by specifying an IP address but not it's DNS you have a DNS problem.

C:\>pathping: Pathping is unique to Window's, and is basically a combination of the Ping and Tracert commands. Pathping traces the route to the destination address then launches a 25 second test of each router along the way, gathering statistics on the rate of data loss along each hop.

C:\>ping: Ping is the most basic TCP/IP command, and it's the same as placing a phone call to your best friend. You pick up your telephone and dial a number, expecting your best friend to reply with "Hello" on the other end. Computers make phone calls to each other over a network by using a Ping command. The Ping commands main purpose is to place a phone call to another computer on the network, and request an answer. Ping has 2 options it can use to place a phone call to another computer on the network. It can use the computers name or IP address.

C:\>route: The route command displays the computers routing table. A typical computer, with a single network interface, connected to a LAN, with a router is fairly simple and generally doesn't pose any network problems. But if you're having trouble accessing other computers on your network, you can use the route command to make sure the entries in the routing table are correct.

C:\>tracert: The tracert command displays a list of all the routers that a packet has to go through to get from the computer where tracert is run to any other computer on the internet.

RECORD	
OBS	
VIVA	
TOTAL	

RESULT:

Thus the above list of primitive has been studied.

DATE: **2.Write a HTTP web client program to download a web page using TCP sockets**

AIM:

To Write a HTTP web client program to download a web page using TCP sockets.

ALGORITHM:

CLIENT SIDE:

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

SERVER SIDE

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

PROGRAM

```
import 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 PROGRAM

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

RECORD	
OBS	
VIVA	
TOTAL	

RESULT:

The webpage is successfully downloaded and the contents are displayed and verified.

DATE: **3. a. SOCKET PROGRAM FOR ECHO**

AIM:

To write a socket program for implementation of echo.

ALGORITHM:

CLIENT SIDE

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

SERVER SIDE

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

PROGRAM:

ECHO CLIENT

```
import java.io.*;
import java.net.*;
public class Eclient{
public static void main(String args[]){
Socket c=null;
String line;
DataInputStream is,is1;
PrintStream os;
try{
c=new Socket("localhost",8080);
}
catch(IOException e){
System.out.println(e);
}
try{
os=new PrintStream(c.getOutputStream());
is=new DataInputStream(System.in);
is1=new DataInputStream(c.getInputStream());
do{
System.out.println("client");
line=is.readLine();
os.println(line);
if(!line.equals("exit"))
System.out.println("server:"+is1.readLine());
```

```
}while(!line.equals("exit"));
}
catch(IOException e)
{
System.out.println("socket closed");
}}}
```

Echo Server:

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

RECORD	
OBS	
VIVA	
TOTAL	

RESULT:

Thus the program for simulation of echo server was written & executed.

DATE: **3.B. CLIENT- SERVER APPLICATION FOR CHAT**

AIM:

To write a client-server application for chat using TCP

ALGORITHM:

CLIENT

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

SERVER

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

PROGRAM:

TCPserver1.java

```
import java.net.*;
import java.io.*;
public class TCPserver1{
 public static void main(String arg[]){
ServerSocket s=null;
String line;
DataInputStream is=null,is1=null;
PrintStream os=null;
Socket c=null;
try{
s=new ServerSocket(9999);
}
catch(IOException e){
System.out.println(e);
}
try{
c=s.accept();
is=new DataInputStream(c.getInputStream());
is1=new DataInputStream(System.in);
os=new PrintStream(c.getOutputStream());
do{
line=is.readLine();
System.out.println("Client:"+line);
```

```
System.out.println("Server:");
line=is1.readLine();
os.println(line);
}
while(line.equalsIgnoreCase("quit")==false);
is.close();
os.close();
}
catch(IOException e){
System.out.println(e);
}}}
```

TCPclient1.java

```
import java.net.*;
import java.io.*;
 public class TCPclient1{
 public static void main(String arg[]){
Socket c=null;
String line;
DataInputStream is,is1;
PrintStream os;
try{
c=new Socket("10.0.200.36",9999);
}
catch(IOException e){
System.out.println(e);
}
try{
 os=new PrintStream(c.getOutputStream());
is=new DataInputStream(System.in);
is1=new DataInputStream(c.getInputStream());
do{
System.out.println("Client:");
line=is.readLine();
os.println(line);
System.out.println("Server:" + is1.readLine());
}
while(line.equalsIgnoreCase("quit")==false);
is1.close();
os.close();
}
catch(IOException e){
System.out.println("Socket Closed!Message Passing is over");
}}
```

RECORD	
OBS	
VIVA	
TOTAL	

RESULT:

Thus the above program a client-server application for chat using TCP / IP was executed and successfully.

DATE: **3.C. FILE TRANSFER IN CLIENT & SERVER**

AIM:

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

ALGORITHM:

CLIENT SIDE

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

SERVER SIDE

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

PROGRAM:

CLIENT SIDE

```
import java.net.*;
import java.io.*;
public class FileClient{
  public static void main (String [] args ) throws IOException { int
    filesize=6022386; // filesize temporary hardcoded long start =
    System.currentTimeMillis();
    int bytesRead;
    int current = 0;
    // localhost for testing
    Socket sock = new Socket("127.0.0.1",13267);
    System.out.println("Connecting...");
    // receive file
    byte [] mybytearray = new byte [filesize];
    InputStream is = sock.getInputStream();
    FileOutputStream fos = new FileOutputStream("source-copy.pdf");
    BufferedOutputStream bos = new BufferedOutputStream(fos);
    bytesRead = is.read(mybytearray,0,mybytearray.length);

    current = bytesRead;
    // thanks to A. Cádiz for the bug fix do {
      bytesRead =
         is.read(mybytearray, current, (mybytearray.length-current));
      if(bytesRead >= 0) current += bytesRead;
    } while(bytesRead > -1);
```

```
   bos.write(mybytearray, 0 , current);
   bos.flush();
   long end = System.currentTimeMillis();
   System.out.println(end-start);
   bos.close();
   sock.close();
 }}
```

SERVER SIDE

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

public class FileServer
{
  public static void main (String [] args ) throws IOException {
    ServerSocket servsock = new ServerSocket(13267); while (true)
    {
     System.out.println("Waiting..."); Socket sock =
     servsock.accept(); System.out.println("Accepted
     connection : " + sock); File myFile = new File
     ("source.pdf");
     byte [] mybytearray = new byte [(int)myFile.length()];
     FileInputStream fis = new FileInputStream(myFile);
     BufferedInputStream bis = new BufferedInputStream(fis);
     bis.read(mybytearray,0,mybytearray.length);
     OutputStream os = sock.getOutputStream();
     System.out.println("Sending...");
     os.write(mybytearray,0,mybytearray.length);
     os.flush();
     sock.close();
     }}}
```

RECORD	
OBS	
VIVA	
TOTAL	

RESULT:

Thus the File transfer Operation is done & executed successfully.

DATE: **4. Simulation of DNS using UDP sockets.**

AIM:

To write a program to Simulation of DNS using UDP sockets..

ALGORITHM:

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

PROGRAM:

// UDP DNS Server-
Udpdnsserver

```
java import java.io.*;
import java.net.*;
public class Udpdnsserver{
private static int indexOf(String[] array, String str){
str = str.trim();
for (int i=0; i < array.length; i++){
if (array[i].equals(str)) return i;
 }
return -1;
 }
public static void main(String arg[])throws IOException{
String[] hosts = {"yahoo.com", "gmail.com","cricinfo.com", "facebook.com"};
String[] ip = {"68.180.206.184", "209.85.148.19","80.168.92.140", "69.63.189.16"};
System.out.println("Press Ctrl + C to Quit");
 while (true){
DatagramSocket serversocket=new DatagramSocket(1362);
byte[] senddata = new byte[1021];
 byte[] receivedata = new byte[1021];
 DatagramPacket recvpack = new DatagramPacket(receivedata, receivedata.length);
 serversocket.receive(recvpack);
String sen = new String(recvpack.getData()); InetAddress
 ipaddress = recvpack.getAddress(); int port = recvpack.getPort();
 String capsent;
 System.out.println("Request for host " + sen);
if(indexOf (hosts, sen) != -1) capsent = ip[indexOf
(hosts, sen)]; else capsent = "Host Not Found";
senddata = capsent.getBytes();
 DatagramPacket pack = new DatagramPacket (senddata, senddata.length,ipaddress,port);
serversocket.send(pack);
serversocket.close();
}}}
```

//UDP DNS Client –Udpdnsclient

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

RECORD	
OBS	
VIVA	
TOTAL	

RESULT:

Thus the above program a client-server application for chat using UDP was executed and successfully

DATE: **5.a.Write a code simulating ARP protocols.**

AIM

To implement Address Resolution Protocol .

ALGORITHM

CLIENT SIDE

1. Establish a connection between the Client and Server. Socket ss=new Socket(InetAddress.getLocalHost(),1100);
2. Create instance output stream writer PrintWriter ps=new PrintWriter(s.getOutputStream(),true);
3. Get the IP Address to resolve its physical address.
4. Send the IPAddress to its output Stream.ps.println(ip);
5. Print the Physical Address received from the server.

SERVER SIDE

1. Accept the connection request by the client. ServerSocket ss=new ServerSocket(2000);Socket s=ss.accept();
2. Get the IPaddress from its inputstream. BufferedReader br1=new BufferedReader(newInputStreamReader(s.getInputStream())); ip=br1.readLine();
3. During runtime execute the processRuntime r=Runtime.getRuntime(); Process p=r.exec("arp -a "+ip);
4. Send the Physical Address to the client.

PROGRAM

ARP CLIENT

```
import java.io.*;
import java.net.*;
class ArpClient{
public static void main(String args[])throws IOException{
try{
Socket ss=new Socket(InetAddress.getLocalHost(),1100);
PrintStream ps=new PrintStream(ss.getOutputStream());
BufferedReader br=new BufferedReader(newInputStreamReader(System.in));
String ip;
System.out.println("Enter the IPADDRESS:");
ip=br.readLine();
ps.println(ip);
String str,data;
BufferedReader br2=new BufferedReader(newInputStreamReader(ss.getInputStream()));
System.out.println("ARP From Server::");
do{
str=br2.readLine();
System.out.println(str);
}
while(!(str.equalsIgnoreCase("end")));
}
```

```
catch(IOException e){
System.out.println("Error"+e);
}}}
```

ARP SERVER

```
import java.io.*;
import java.net.*;
class ArpServer{
public static void main(String args[])throws IOException{
try{
ServerSocket ss=new ServerSocket(1100);
Socket s=ss.accept();
PrintStream ps=new PrintStream(s.getOutputStream());
BufferedReader br1=new BufferedReader(newInputStreamReader(s.getInputStream()));
String ip;
 ip=br1.readLine();
Runtime r=Runtime.getRuntime();
Process p=r.exec("arp -a "+ip);
BufferedReader br2=new BufferedReader(newInputStreamReader(p.getInputStream()));
String str;
while((str=br2.readLine())!=null){
ps.println(str);
}}
catch(IOException e){
System.out.println("Error"+e); }}}
```

RECORD	
OBS	
VIVA	
TOTAL	

RESULT

Thus the implementation of ARP is done & executed successfully.

DATE: **5.b.Write a code simulating RARP protocols.**

AIM:

To write a java program for simulating RARP protocols.

ALGORITHM:

CLIENT

1.Start the program
2. using datagram sockets UDP function is established.
2.Get the MAC address to be converted into IP address.
3.Send this MAC address to server.
4.Server returns the IP address to client.

SERVER

1. Start the program.
2. Server maintains the table in which IP and corresponding MAC addresses are stored.
3. Read the MAC address which is send by the client.
4. Map the IP address with its MAC address and return the IP address to client.

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=newDatagramPacket(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();
        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(inti=0;i<ip.length;i++){
        if(s.equals(mac[i])){
         sendbyte=ip[i].getBytes();
        DatagramPacket  sender=newDatagramPacket(sendbyte,sendbyte.length,addr,port);
        server.send(sender);
        break;
        }}
        break;
        }}
        catch(Exception e){
                System.out.println(e);
}}}
```

RECORD	
OBS	
VIVA	
TOTAL	

RESULT:

Thus the implementation of RARP is done & executed successfully.