Department of Information Technology
Anna University, MIT Campus

CS9306

Computer Networks Laboratory

LAB MANUAL

AIM:

To have hands-on experience in network programming and to use simulation tools to analyse network protocols.

PROGRAMME EDUCATIONAL OBJECTIVES:

- To learn socket programming
- To use simulation tools.
- To analyse the performance of protocols in different layers in computer networks using simulation tools.

EXPERIMENTS:

1. Applications using TCP Sockets like
    a. Echo client and echo server
    b. File transfer
    c. Remote command execution
    d. Chat
    e. Concurrent server
2. Applications using UDP Sockets like
    a. DNS
    b. SNMP
3. Applications using Raw Sockets like
    a. Ping
    b. Traceroute
4. RPC
5. Experiments using simulators like OPNET:
    a. Performance comparison of MAC protocols
    b. Performance comparison of routing protocols
    c. Study of TCP/UDP performance

OUTCOMES:

To students should be able to understand and implement most of the protocols and able to simulate the sample network models using simulators.

LIST OF EXPERIMENTS

I. JAVA PROGRAMMING

1. Implement - echo client and echo server application using TCP Sockets
2. Implement - echo client and echo server application using UDP Sockets
3. Implement- File Transfer Protocol using TCP Sockets
4. Implement - chat application using TCP Sockets
5. Implement - concurrent server communication using TCP Sockets
6. Implement - Ping command using Raw Sockets
7. Implement - Trace route application using Raw Sockets
8. Implement - Remote command execution using TCP Sockets
9. Implement - DNS using UDP Sockets
10. Implement - Sliding window protocol using TCP & UDP socket programming
11. Implement – CRC error detection code for given message
12. Implement – SNMP protocol using socket programming

II. Network simulator tutorial

1. Generating network topology using NS2
2. UDP Data traffic
3. TCP Data traffic
4. Wireless scenario
5. Topology and movement generation
6. Energy model
7. Wired cum wireless scenario
8. Analyse the performance of a network using xgraph

STUDY OF BASIC NETWORK COMMANDS

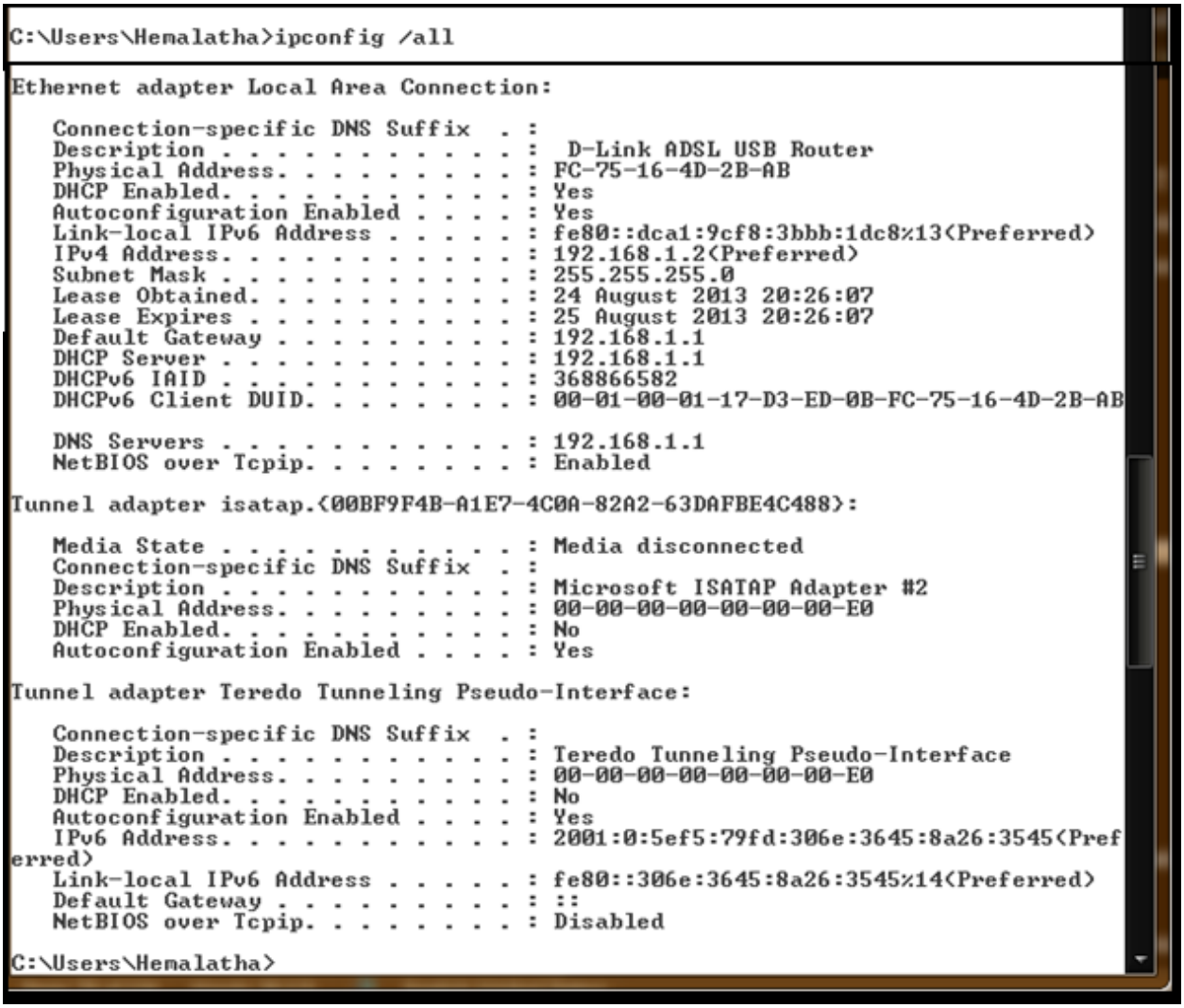
Aim:

Introduction to basic network commands

1. Ip configuration

Use:

Displays all current TCP/IP network configuration values and refreshes Dynamic Host Configuration Protocol (DHCP) and Domain Name System (DNS) settings. Used without parameters, **ipconfig** displays the IP address, subnet mask, and default gateway for all adapters.



```
C:\Users\Hemalatha>ipconfig /all

Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . :  D-Link ADSL USB Router
   Physical Address. . . . . . . . . : FC-75-16-4D-2B-AB
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::dca1:9cf8:3bbb:1dc8%13(Preferred)
   IPv4 Address. . . . . . . . . . . : 192.168.1.2(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Lease Obtained. . . . . . . . . . : 24 August 2013 20:26:07
   Lease Expires . . . . . . . . . . : 25 August 2013 20:26:07
   Default Gateway . . . . . . . . . : 192.168.1.1
   DHCP Server . . . . . . . . . . . : 192.168.1.1
   DHCPv6 IAID . . . . . . . . . . . : 368866582
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-17-D3-ED-0B-FC-75-16-4D-2B-AB

   DNS Servers . . . . . . . . . . . : 192.168.1.1
   NetBIOS over Tcpip. . . . . . . . : Enabled

Tunnel adapter isatap.{00BF9F4B-A1E7-4C0A-82A2-63DAFBE4C488}:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Microsoft ISATAP Adapter #2
   Physical Address. . . . . . . . . : 00-00-00-00-00-00-00-E0
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes

Tunnel adapter Teredo Tunneling Pseudo-Interface:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Teredo Tunneling Pseudo-Interface
   Physical Address. . . . . . . . . : 00-00-00-00-00-00-00-E0
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
   IPv6 Address. . . . . . . . . . . : 2001:0:5ef5:79fd:306e:3645:8a26:3545(Pref
erred)
   Link-local IPv6 Address . . . . . : fe80::306e:3645:8a26:3545%14(Preferred)
   Default Gateway . . . . . . . . . : ::
   NetBIOS over Tcpip. . . . . . . . : Disabled

C:\Users\Hemalatha>
```

2. Ping command

Use:

If you are having connectivity problems, you can use the **ping** command to check the destination IP address you want to reach and record the results. The **ping** command displays whether the destination responded and how long it took to receive a reply. If there is an error in the delivery to the destination, the **ping** command displays an error message.

```
C:\Users\Admin>ping 192.168.100.1

Pinging 192.168.100.1 with 32 bytes of data:
Reply from 192.168.100.1: bytes=32 time=316ms TTL=255
Reply from 192.168.100.1: bytes=32 time=82ms TTL=255
Reply from 192.168.100.1: bytes=32 time=306ms TTL=255
Reply from 192.168.100.1: bytes=32 time=7ms TTL=255

Ping statistics for 192.168.100.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 7ms, Maximum = 316ms, Average = 177ms
```

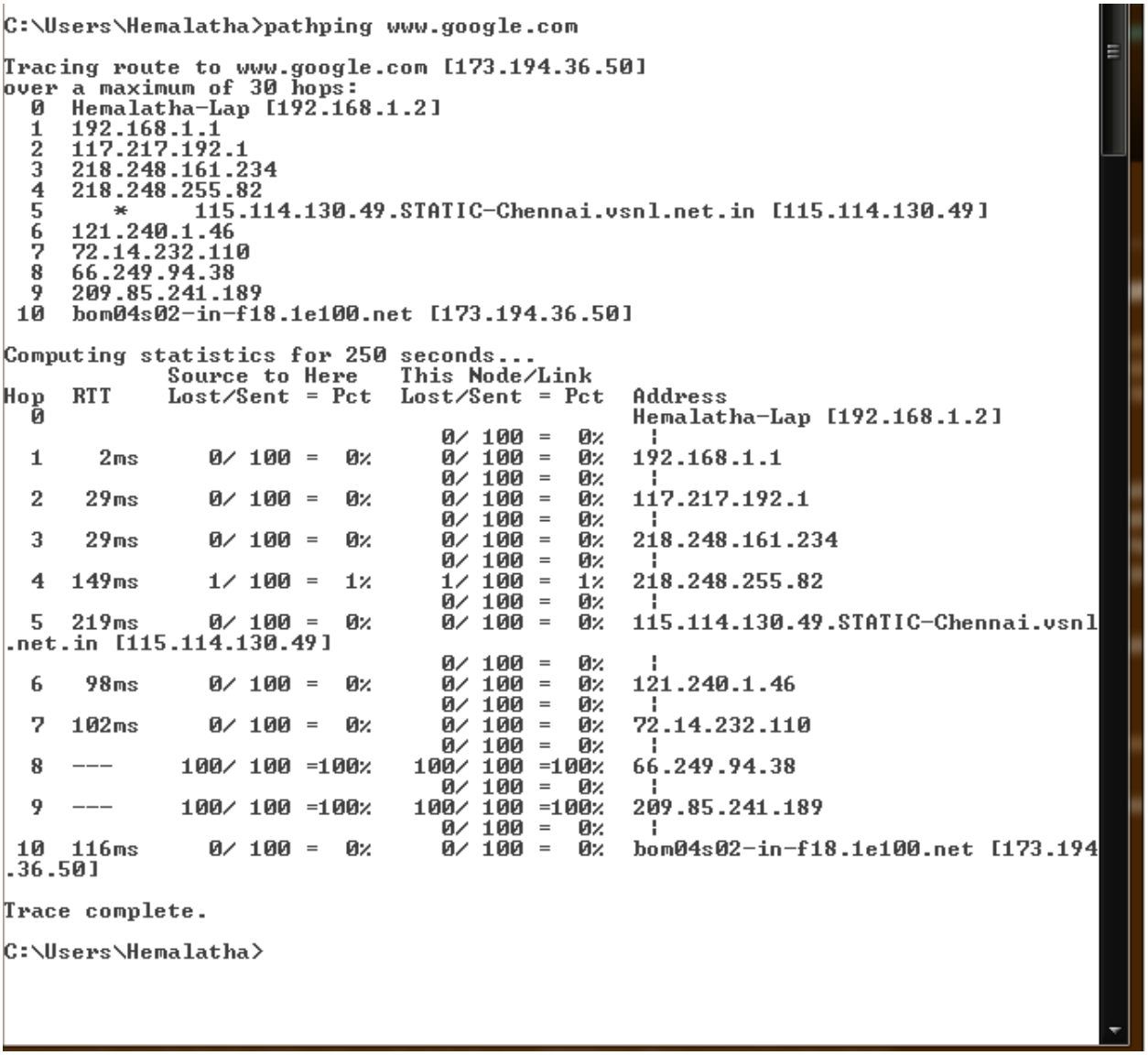
3. Trace route command

Use:

If you are having connectivity problems, you can use the **ping** command to check the destination IP address you want to reach and record the results. The **ping** command displays whether the destination responded and how long it took to receive a reply. If there is an error in the delivery to the destination, the **ping** command displays an error message.

4. Pathping command

The **pathping** command is a route tracing tool that combines features of the **ping** and **tracert** commands with additional information that neither of those tools provides. The **pathping** command sends packets to each router on the way to a final destination over a period of time, and then computes results based on the packets returned from each hop. Since the command shows the degree of packet loss at any given router or link, it is easy to determine which routers or links might be causing network problems.



```
C:\Users\Hemalatha>pathping www.google.com

Tracing route to www.google.com [173.194.36.50]
over a maximum of 30 hops:
  0  Hemalatha-Lap [192.168.1.2]
  1  192.168.1.1
  2  117.217.192.1
  3  218.248.161.234
  4  218.248.255.82
  5     *     115.114.130.49.STATIC-Chennai.vsnl.net.in [115.114.130.49]
  6  121.240.1.46
  7  72.14.232.110
  8  66.249.94.38
  9  209.85.241.189
 10  bom04s02-in-f18.1e100.net [173.194.36.50]

Computing statistics for 250 seconds...
            Source to Here   This Node/Link
Hop  RTT    Lost/Sent = Pct  Lost/Sent = Pct  Address
  0                                           Hemalatha-Lap [192.168.1.2]
                                     0/ 100 =  0%   |
  1    2ms     0/ 100 =  0%        0/ 100 =  0%  192.168.1.1
                                     0/ 100 =  0%   |
  2   29ms     0/ 100 =  0%        0/ 100 =  0%  117.217.192.1
                                     0/ 100 =  0%   |
  3   29ms     0/ 100 =  0%        0/ 100 =  0%  218.248.161.234
                                     0/ 100 =  0%   |
  4  149ms     1/ 100 =  1%        1/ 100 =  1%  218.248.255.82
                                     0/ 100 =  0%   |
  5  219ms     0/ 100 =  0%        0/ 100 =  0%  115.114.130.49.STATIC-Chennai.vsnl
.net.in [115.114.130.49]
                                     0/ 100 =  0%   |
  6   98ms     0/ 100 =  0%        0/ 100 =  0%  121.240.1.46
                                     0/ 100 =  0%   |
  7  102ms     0/ 100 =  0%        0/ 100 =  0%  72.14.232.110
                                     0/ 100 =  0%   |
  8  ---     100/ 100 =100%      100/ 100 =100%  66.249.94.38
                                     0/ 100 =  0%   |
  9  ---     100/ 100 =100%      100/ 100 =100%  209.85.241.189
                                     0/ 100 =  0%   |
 10  116ms     0/ 100 =  0%        0/ 100 =  0%  bom04s02-in-f18.1e100.net [173.194
.36.50]

Trace complete.

C:\Users\Hemalatha>
```

5. arp command

The address resolution protocol (arp) is a protocol used by the Internet Protocol (IP) , specifically IPv4, to map IP network addresses to the hardware addresses used by a data link protocol. The protocol operates below the network layer as a part of the interface between the OSI network and OSI link layer. It is used when IPv4 is used over Ethernet.

The term address resolution refers to the process of finding an address of a computer in a network. The address is "resolved" using a protocol in which a piece of information is sent by a client process executing on the local computer to a server process executing on a remote computer. The information received by the server allows the server to uniquely identify the network system for which the

address was required and therefore to provide the required address. The address resolution procedure is completed when the client receives a response from the server containing the required address.

```
C:\Users\Admin>arp -a

Interface: 192.168.101.64 --- 0xa
  Internet Address      Physical Address      Type
  192.168.101.1         00-20-9c-69-8c-00     dynamic
  192.168.101.47        00-50-c2-c5-1b-30     dynamic
  192.168.101.54        00-24-8c-40-b3-c3     dynamic
  192.168.101.55        d0-27-88-38-02-9e     dynamic
  192.168.101.66        10-bf-48-08-ac-85     dynamic
  192.168.101.73        28-92-4a-4c-b2-06     dynamic
  192.168.101.75        ec-a8-6b-23-ff-1b     dynamic
  192.168.101.77        28-92-4a-4d-b8-2e     dynamic
  192.168.101.82        d0-27-88-38-02-02     dynamic
  192.168.101.84        00-25-64-e9-2e-33     dynamic
  192.168.101.87        d0-27-88-3d-a2-a4     dynamic
  192.168.101.90        54-53-ed-2d-41-43     dynamic
  192.168.101.96        00-16-e6-9d-d9-c6     dynamic
  192.168.101.101       00-26-9e-d5-06-dc     dynamic
  192.168.101.255       ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static

C:\Users\Admin>_
```

6. hostname command

Display the hostname of the machine the command is being run on. Additional information about the term hostname can be found on our hostname dictionary definition.

```
C:\Users\Admin>hostname
Sowmiya-PC
```

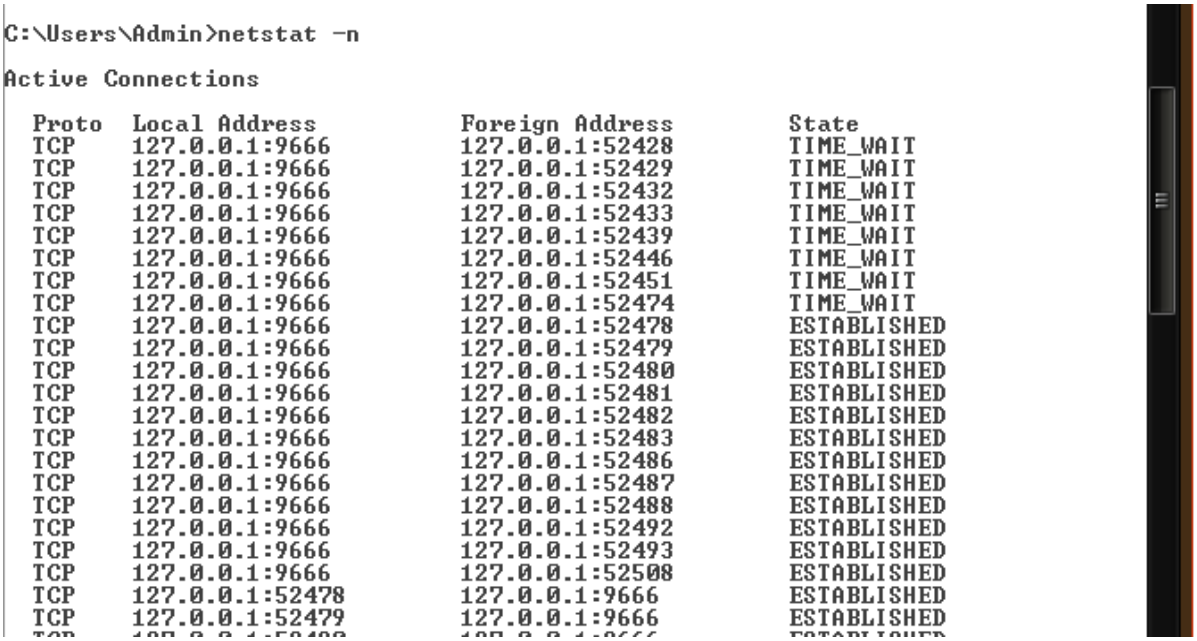
7. netstat command

The netstat command is used to display the TCP/IP network protocol statistics and information.

NETSTAT [-a] [-e] [-n] [-s] [-p proto] [-r] [interval]

-a	Displays all connections and listening ports.
-e	Displays Ethernet statistics. This may be combined with the -s option.

-n	Displays addresses and port numbers in numerical form.
-p	proto Shows connections for the protocol specified by proto; proto may be TCP or UDP. If used with the -s option to display per-protocol statistics, proto may beTCP, UDP, or IP.
-r	Displays the routing table.
-s	Displays per-protocol statistics. By default, statistics are shown for TCP, UDP and IP; the -p option may be used to specify a subset of the default.
interval	Redisplays selected statistics, pausing interval seconds between each display. Press CTRL+C to stop redisplaying statistics. If omitted, netstat will print the current configuration information once.



```
C:\Users\Admin>netstat -n

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    127.0.0.1:9666         127.0.0.1:52428        TIME_WAIT
  TCP    127.0.0.1:9666         127.0.0.1:52429        TIME_WAIT
  TCP    127.0.0.1:9666         127.0.0.1:52432        TIME_WAIT
  TCP    127.0.0.1:9666         127.0.0.1:52433        TIME_WAIT
  TCP    127.0.0.1:9666         127.0.0.1:52439        TIME_WAIT
  TCP    127.0.0.1:9666         127.0.0.1:52446        TIME_WAIT
  TCP    127.0.0.1:9666         127.0.0.1:52451        TIME_WAIT
  TCP    127.0.0.1:9666         127.0.0.1:52474        TIME_WAIT
  TCP    127.0.0.1:9666         127.0.0.1:52478        ESTABLISHED
  TCP    127.0.0.1:9666         127.0.0.1:52479        ESTABLISHED
  TCP    127.0.0.1:9666         127.0.0.1:52480        ESTABLISHED
  TCP    127.0.0.1:9666         127.0.0.1:52481        ESTABLISHED
  TCP    127.0.0.1:9666         127.0.0.1:52482        ESTABLISHED
  TCP    127.0.0.1:9666         127.0.0.1:52483        ESTABLISHED
  TCP    127.0.0.1:9666         127.0.0.1:52486        ESTABLISHED
  TCP    127.0.0.1:9666         127.0.0.1:52487        ESTABLISHED
  TCP    127.0.0.1:9666         127.0.0.1:52488        ESTABLISHED
  TCP    127.0.0.1:9666         127.0.0.1:52492        ESTABLISHED
  TCP    127.0.0.1:9666         127.0.0.1:52493        ESTABLISHED
  TCP    127.0.0.1:9666         127.0.0.1:52508        ESTABLISHED
  TCP    127.0.0.1:52478        127.0.0.1:9666         ESTABLISHED
  TCP    127.0.0.1:52479        127.0.0.1:9666         ESTABLISHED
```

```
C:\Users\Admin>netstat -s

IPv4 Statistics

  Packets Received                   = 165194
  Received Header Errors             = 2
  Received Address Errors            = 2737
  Datagrams Forwarded                = 0
  Unknown Protocols Received         = 0
  Received Packets Discarded         = 11021
  Received Packets Delivered         = 151858
  Output Requests                    = 115496
  Routing Discards                   = 0
  Discarded Output Packets           = 10
  Output Packet No Route             = 3
  Reassembly Required                = 0
  Reassembly Successful              = 0
  Reassembly Failures                = 0
  Datagrams Successfully Fragmented  = 0
  Datagrams Failing Fragmentation    = 0
  Fragments Created                  = 0

IPv6 Statistics

  Packets Received                   = 26812
  Received Header Errors             = 0
  Received Address Errors            = 520
  Datagrams Forwarded                = 0
  Unknown Protocols Received         = 0
  Received Packets Discarded         = 4628
  Received Packets Delivered         = 21818
  Output Requests                    = 627
  Routing Discards                   = 0
  Discarded Output Packets           = 0
  Output Packet No Route             = 6
  Reassembly Required                = 0
  Reassembly Successful              = 0
```

8. route command

Command to manually configure the routes in the routing table.

ROUTE [-f] [-p] [command [destination] [MASK netmask] [gateway] [METRIC metric] [IF interface]

-f	Clears the routing tables of all gateway entries. If this is used in conjunction with one of the commands, the tables are cleared prior to running the command.
-p	When used with the ADD command, makes a route persistent across boots of the system. By default, routes are not preserved when the system is restarted. When used with the PRINT command, displays the list of registered persistent routes. Ignored for all other commands, which always affect the appropriate persistent routes. This option is not supported Windows'95. command
command	One of these:

	PRINT Prints a route ADD Adds a route DELETE Deletes a route CHANGE Modifies an existing route destination
destination	Specifies the host.
MASK	Specifies that the next parameter is the 'netmask' value.
netmask	Specifies a subnet mask value for this route entry. If not specified, it defaults to 255.255.255.255.
gateway	Specifies gateway.
interface	the interface number for the specified route.
METRIC	Specifies the metric, ie. cost for the destination.

```
C:\Users\Admin>route PRINT
===========================================================================
Interface List
 10...d0 27 88 3d 9d 66 ......Realtek PCIe GBE Family Controller
  1...........................Software Loopback Interface 1
 11...00 00 00 00 00 00 00 e0 Microsoft ISATAP Adapter
 12...00 00 00 00 00 00 00 e0 Teredo Tunneling Pseudo-Interface
===========================================================================

IPv4 Route Table
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0    192.168.101.1   192.168.101.64     20
        127.0.0.0        255.0.0.0         On-link         127.0.0.1    306
        127.0.0.1  255.255.255.255         On-link         127.0.0.1    306
  127.255.255.255  255.255.255.255         On-link         127.0.0.1    306
    192.168.101.0    255.255.255.0         On-link    192.168.101.64    276
   192.168.101.64  255.255.255.255         On-link    192.168.101.64    276
  192.168.101.255  255.255.255.255         On-link    192.168.101.64    276
        224.0.0.0        240.0.0.0         On-link         127.0.0.1    306
        224.0.0.0        240.0.0.0         On-link    192.168.101.64    276
  255.255.255.255  255.255.255.255         On-link         127.0.0.1    306
  255.255.255.255  255.255.255.255         On-link    192.168.101.64    276
===========================================================================
Persistent Routes:
  None

IPv6 Route Table
===========================================================================
Active Routes:
 If Metric Network Destination      Gateway
 12     58 ::/0                     On-link
  1    306 ::1/128                  On-link
 12     58 2001::/32                On-link
 12    306 2001:0:5ef5:79fd:2083:2bce:3f57:9abf/128
                                    On-link
 10    276 fe80::/64                On-link
 12    306 fe80::/64                On-link
 12    306 fe80::2083:2bce:3f57:9abf/128
                                    On-link
 10    276 fe80::7574:2df1:19bb:2d02/128
                                    On-link
  1    306 ff00::/8                 On-link
 12    306 ff00::/8                 On-link
 10    276 ff00::/8                 On-link
===========================================================================
Persistent Routes:
  None

C:\Users\Admin>
```

9. nslookup command

MS-DOS utility that enables a user to look up an IP address of a domain or host on a network.

```
C:\Users\Admin>nslookup
120.1.168.192.in-addr.arpa
        primary name server = localhost
        responsible mail addr = nobody.invalid
        serial  = 1
        refresh = 600 (10 mins)
        retry   = 1200 (20 mins)
        expire  = 604800 (7 days)
        default TTL = 10800 (3 hours)
Default Server:  UnKnown
Address:  192.168.1.120

> www.google.com
Server:  UnKnown
Address:  192.168.1.120

Non-authoritative answer:
Name:    www.google.com
Addresses:  2404:6800:4009:803::1014
          173.194.36.52
          173.194.36.48
          173.194.36.49
          173.194.36.50
          173.194.36.51

> www.facebook.com
Server:  UnKnown
Address:  192.168.1.120
```

10. nbtstat command

Displays NetBIOS over TCP/IP (NetBT) protocol statistics, NetBIOS name tables for both the local computer and remote computers, and the NetBIOS name cache. Nbtstat allows a refresh of the NetBIOS name cache and the names registered with Windows Internet Name Service (WINS).

```
C:\Users\Admin>nbtstat -n

Local Area Connection:
Node IpAddress: [192.168.101.64] Scope Id: []

                NetBIOS Local Name Table

       Name               Type         Status
    ---------------------------------------------
    SOWMIYA-PC     <00>  UNIQUE      Registered
    WORKGROUP      <00>  GROUP       Registered
    SOWMIYA-PC     <20>  UNIQUE      Registered
    WORKGROUP      <1E>  GROUP       Registered
```

Ex.NO:1 IMPLEMENT - ECHO SERVER AND CLIENT USING SOCKET PROGRAMING

AIM

To write a java program for implementing echoclient and echoserver.

THEORY:

The server does is wait for a connection, then uses the Socket produced by that connection to create an InputStream and OutputStream. After that, everything it reads from the InputStream it echoes to the OutputStream until it receives the line END, at which time it closes the connection.

The client makes the connection to the server, then creates an OutputStream. Lines of text are sent through the OutputStream. The client also creates an InputStream to hear what the server is saying (which, in this case, is just the words echoed back). Both the server and client use the same port number and the client uses the local loopback address to connect to the server on the same machine so you don't have to test it over a network.

You can see that the ServerSocket just needs a port number, not an IP address (since it's running on this machine!). When you call accept( ), the method blocks until some client tries to connect to it. That is, it's there waiting for a connection but other processes can run . When a connection is made, accept( ) returns with a Socket object representing that connection.

If the ServerSocket constructor fails, the program just quits (notice we must assume that the constructor for ServerSocket doesn't leave any open network sockets lying around if it fails). For this case, main( ) throws IOException so a try block is not necessary.

If accept( ) fails, then we must assume that the Socket doesn't exist or hold any resources, so it doesn't need to be cleaned up. If it's successful, however, the following statements must be in a try-finally block so that if they fail the Socket will still be cleaned up.

ALGORITHM

SERVER SIDE

1. Create the server socket and begin listening.
2. Call the accept() method to get new connections.
3. Create input and output streams for the returned socket.
4. Conduct the conversation based on the agreed protocol.
5. Close the client streams and socket.

6. Go back to step 2 or continue to step 7.
7. Close the server socket.

CLIENT SIDE

1. Create the client socket connection.
2. Acquire read and write streams to the socket.

PROGRAM

ECHOSERVER

```
import java.io.*;
import java.net.*;
classEchoServer
{
publicEchoServer(intportnum)
{
try
{
server=new ServerSocket(portnum);
}
catch(Exception err)
{
System.out.println(err);
}
}
public void serve()
{
try
{
while(true)
{
Socket client=server.accept();
BufferedReader r=new BufferedReader(new
InputStreamReader(client.getInputStream()));
PrintWriter w=new PrintWriter(client.getOutputStream(),true);
w.println("Welcome to the Java EchoServer. Type'bye' to close.");
String line;
do
{
line=r.readLine();
```

```
if(line !=null)
System.err.println("server got "+line+" from client");
w.println("Got:"+line);
}
while( !line.trim().equals("bye"));
client.close();
}
}
catch(Exception err)
{
System.err.println(err);
}
}
public static void main(String[] args)
{
EchoServer s=new EchoServer(9999);
s.serve();
}
privateServerSocket server;
}
```

ECHOCLIENT

```
import java.io.*;
import java.net.*;
classEchoClient
{
public static void main(String[] args)
{
try
{
Socket s=new Socket("127.0.0.1",9999);
BufferedReader r=new BufferedReader(new InputStreamReader(s.getInputStream()));
PrintWriter w=new PrintWriter(s.getOutputStream(),true);
BufferedReader con=new BufferedReader(new InputStreamReader(System.in));
String line;
do
{
line=r.readLine();
if(line !=null)
System.out.println(line);
line=con.readLine();
w.println(line);
```

```
}
while( !line.trim().equals("bye"));
}
catch(Exception err)
{
System.err.println(err);
}
}
}
```

OUTPUT:

SERVER:
Message from Client:Hi
Echoing the message :
HiMessage from Client:Computer networks
Echoing the message :
Computer networks

CLIENT:
My msg:
Hi
Echo from Server : Hi
My msg:
Computer networks
Echo from Server : Computer networks
My msg:
bye

RESULT

Thus the above program for echoclient&echoserver using socket program has been executed successfully.

EX 2 : IMPLEMENT - ECHO CLIENT AND ECHO SERVER APPLICATION USING UDP SOCKETS

AIM:

To write a java program to implement echo client and echo server application using TCP sockets.

THEORY:

Receiving Datagrams

The DatagramPacket class is used to receive and send data over DatagramSocket classes. The packet class contains connection information as well as the data. As was explained earlier, datagrams are self-contained transmission units. The DatagramPacket class encapsulates these units. The following lines receive data from a datagram socket:

```
DatagramPacket packet = new DatagramPacket(new byte[512], 512);
clientSocket.receive(packet);
```

The constructor for the packet must know where to place the received data. A 512-byte buffer is created and passed to the constructor as the first parameter. The second constructor parameter is the size of the buffer. Like the accept() method in the ServerSocket class, the receive() method blocks until data is available.

Sending Datagrams

Sending datagrams is really very simple; all that's needed is a complete address. Addresses are created and tracked using the InetAddress class. This class has no public constructors, but it does contain several static methods that can be used to create an instance of the class. The following list shows the public methods that create InetAddress class instances:

Public InetAddress Creation Methods
InetAddress getByName(String host);
InetAddress[] getAllByName(String host);
InetAddress getLocalHost();

Getting the local host is useful for informational purposes, but only the first two methods are actually used to send packets. Both getByName() and getAllByName() require the name of the destination host. The first method merely returns the first match it finds. The second method is needed because a computer can have more than one address. When this occurs, the computer is said to be *multi-homed*. The computer has one name, but multiple ways to reach it.

ALGORITHM:

SERVER:

STEP 1: Start
STEP 2: Declare the variables for the socket
STEP 3: Specify the family, protocol, IP address and port number
STEP 4: Create a socket using socket() function
STEP 5: Bind the IP address and Port number
STEP 6: Listen and accept the client's request for the connection
STEP 7: Read and Display the client's message
STEP 8: Stop

CLIENT:

STEP 1: Start
STEP 2: Declare the variables for the socket
STEP 3: Specify the family, protocol, IP address and port number
STEP 4: Create a socket using socket() function
STEP 5: Call the connect() function
STEP 6: Read the input message
STEP 7: Send the input message to the server
STEP 8: Display the server's echo
STEP 9: Close the socket
STEP 10: Stop

CLIENT :

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

class UDPClient
{
public static void main(String[] args)
{
 try
{
DatagramSocket dsclient=new DatagramSocket();
InetAddress ipaddr=InetAddress.getByName("localhost");
Scanner in=new Scanner(System.in);
//BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
byte[] senddata=new byte[1024];
byte[] recdata=new byte[1024];
while(true)
{
```

```
System.out.print("Enter the data :");
String tosend=in.nextLine();
//String tosend=br.readLine();
if(tosend.equalsIgnoreCase("bye"))
break;
senddata=tosend.getBytes();
DatagramPacket sendpacket=new
DatagramPacket(senddata,senddata.length,ipaddr,1544);
dsclient.send(sendpacket);
DatagramPacket recpacket=new DatagramPacket(recdata,recdata.length);
dsclient.receive(recpacket);
String rec_frm_ser=new String(recpacket.getData());
System.out.println("Msg. From Server :"+rec_frm_ser);
}
dsclient.close();
}catch(Exception x)
{
x.printStackTrace();
}
}
}
```

SERVER:

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

class UDPServer
{
public static void main(String[] args)
{
try
{
DatagramSocket dsserver=new DatagramSocket(1544);
byte[] sendata=new byte[1024];
byte[] recdata=new byte[1024];
while(true)
{
recdata=new byte[1024];
DatagramPacket recpacket=new DatagramPacket(recdata,recdata.length);
dsserver.receive(recpacket);
String clismsg=new String(recpacket.getData());
if(clismsg.equalsIgnoreCase("bye"))
{
```

```
break;
}
InetAddress ipaddr=recpacket.getAddress();
int port=recpacket.getPort();
String toupcase=clismsg.toUpperCase();
System.out.println("Client's msg :"+clismsg);
sendata=toupcase.getBytes();
DatagramPacket sendpacket=new DatagramPacket(sendata,sendata.length,ipaddr,port);
dsserver.send(sendpacket);
System.out.println("Msg. to Client :"+toupcase);
}
dsserver.close();
}catch(Exception x)
{
x.printStackTrace();
}
}
}
```

OUTPUT:
SERVER:
Client's msg :Hi
Msg. to Client :HI
Client's msg :Hello
Msg. to Client :HELLO
Client's msg :Networks lab
Msg. to Client :NETWORKS LAB

CLIENT:
Enter the data :Hi
Msg. From Server :HI
Enter the data :Hello
Msg. From Server :HELLO
Enter the data :Networks lab
Msg. From Server :NETWORKS LAB
Enter the data :bye

RESULT:

Thus the program for UDP echo client server was executed and the output was verified.

EX 3 : IMPLEMENT - FILE TRANSFER PROTOCOL USING TCP SOCKETS

AIM:

To write a java program to implement the file transfer protocol.

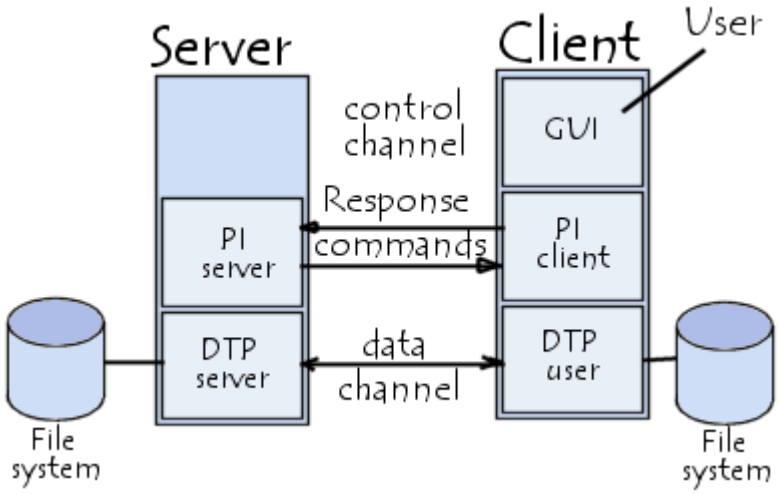
THEORY:

FTP protocol (*File Transfer Protocol*) is, as its name indicates a protocol for transferring files. FTP protocol defines the way in which data must be transferred over a TCP/IP network. The aim of FTP protocol is to:

- allow file sharing between remote machines
- allow independence between client and server machine system files
- enable efficient data transfer

FTP protocol falls within a client-server model, i.e. one machine sends orders (the client) and the other awaits requests to carry out actions (the server). During an FTP connection, two transmission channels are open:

- A channel for commands (control channel)
- A channel for data



So, both the client and server have two processes allowing these two types of information to be managed:

- **DTP** (*Data Transfer Process*) is the process in charge of establishing the connection and managing the data channel. The server side DTP is called *SERVER-DTP*, the client side DTP is called *USER-DTP*

- **PI** (*Protocol Interpreter*) interprets the protocol allowing the DTP to be controlled using commands received over the control channel. It is different on the client and the server:
    - The SERVER-PI is responsible for listening to the commands coming from a USER-PI over the control channel on a data port, establishing the connection for the control channel, receiving FTP commands from the USER-PI over this, responding to them and running the SERVER-DTP.
    - The USER-PI is responsible for establishing the connection with the FTP server, sending FTP commands, receiving responses from the SERVER-PI and controlling the USER-DTP if needed.

When an FTP client is connected to a FTP server, the USER-PI initiates the connection to the server according to the Telnet protocol. The client sends FTP commands to the server, the server interprets them, runs its DTP, then sends a standard response. Once the connection is established, the server-PI gives the port on which data will be sent to the Client DTP. The client DTP then listens on the specified port for data coming from the server.
It is important to note that since the control and data ports are separate channels, it is possible to send commands from one machine and receive data on another. So, for example it is possible to transfer data between FTP servers by passing through a client to send control instructions and by transferring information between two server processes connected on the right port. In this configuration, the protocol imposes that the control channels remain open throughout the data transfer. So a server can stop a transmission if the control channel is broken during transmission.

ALGORITHM

CLIENT:
1. Start the program.
2. Create the client packet.
3. After transferring the packet statementis displayed.
4. Stop the program.

SERVER:
1. Start the program.
2. Create the server socket.
3. Call the I/O stream.
4. Print the file has been sent.

5. Send the intimation to the client.
6. Stop the program

PROGRAM:

SERVER :

```
import java.net.*;
import java.io.*;
import java.util.*;
class FileTransfer
{
Socket soc;
ServerSocket ss;
DataInputStream din;
DataOutputStream dout;
String fname;
File recfile;
String filecon;
FileTransfer() throws Exception
{
ss=new ServerSocket(2788);
soc=ss.accept();
din=new DataInputStream(soc.getInputStream());
dout=new DataOutputStream(soc.getOutputStream());
}
void sendfile() throws Exception
{
Scanner in=new Scanner(System.in);
//dout.writeUTF("R");
//System.out.println("File Name :");
fname=din.readUTF();
System.out.println("File name :"+fname);
recfile=new File(fname);
if(recfile.exists())
{
dout.writeUTF("Y");
if((din.readUTF()).equalsIgnoreCase("Y"))
startsend();
else
{
}
}
else
{
```

```
System.out.println(" Not Found");
dout.writeUTF("N");
}

}
void startsend() throws Exception
{
String filecon;
FileInputStream fos=new FileInputStream(recfile);
int ch;
System.out.println("Transferring File ....");
while(true)
{
ch=fos.read();
//ch=Integer.parseInt(filecon);
if(ch!=-1)
{
dout.writeUTF(String.valueOf(ch));
}
else
break;
}
System.out.println("File Transferring Completed .");
}
}
public class FTPServer
{
public static void main(String[] args) throws Exception
{
FileTransfer fs=new FileTransfer();
fs.sendfile();
System.out.println("Disconnected .");
}
}
```

CLIENT:

```
import java.net.*;
import java.io.*;
import java.util.*;
class FileTransfer
{
Socket soc;
```

```
DataInputStream din;
DataOutputStream dout;
String fname;
File recfile;
String filecon;
FileTransfer() throws Exception
{
soc=new Socket("127.0.0.1",2788);
din=new DataInputStream(soc.getInputStream());
dout=new DataOutputStream(soc.getOutputStream());
}
void rec_file() throws Exception
{
Scanner in=new Scanner(System.in);
//dout.writeUTF("R");
System.out.println("File Name :");
fname=in.nextLine();
dout.writeUTF(fname);
String server_msg=din.readUTF();
if(server_msg.equalsIgnoreCase("Y"))
{
recfile=new File(fname);
if(recfile.exists())
{
System.out.println("File Already exists.Want to replace?(Y/N)");
if((in.nextLine()).equalsIgnoreCase("Y"))
{
dout.writeUTF("Y");
startrec();
}
else
{
System.out.println("Want to give a new File name or exit ?(Y/N)");
if((in.nextLine()).equalsIgnoreCase("Y"))
{
System.out.println("Enter :");
String newname=in.nextLine();
recfile=new File(newname);
dout.writeUTF("Y");
startrec();
}
else
```

```
{
dout.writeUTF("N");
}
}
}
else
{
System.out.println(" Transferring File...");
startrec();
}
}
else
{
System.out.println("File not Found in Server");
}
}
void startrec()
{
try
{
String filecon;
FileOutputStream fos=new FileOutputStream(recfile);
int ch;
System.out.println("Transferring File ....");
while(true)
{
filecon=din.readUTF();
ch=Integer.parseInt(filecon);
if(ch!=-1)
{
fos.write(ch);
 }
else
break;
}
System.out.println("File is Received .");
}catch(Exception x)
{
}
}
}
public class FTPClient
{
```

```
public static void main(String[] args) throws Exception
{
FileTransfer fs=new FileTransfer();
fs.rec_file();
System.out.println("FTP Session Ended .");
}
}
```

OUTPUT:

```
FTP CLIENT
File Name :
C:\Users\Hemalatha\Desktop\input.txt
File Already exists.Want to replace?(Y/N)
N
Want to give a new File name or exit ?(Y/N)
Y
Enter :
output.txt
Transferring File ....
FTP Session Ended .

FTP SERVER
File name :C:\Users\Hemalatha\Desktop\input.txt
Transferring File ....
File Transferring Completed .
Disconnected .
```

RESULT:

Thus the implementation of FTP client and server has been executed and output has been verified successfully.

EX NO: 4 IMPLEMENT - CHAT APPLICATION USING TCP SOCKETS

AIM:

To write a java program to implement chat client and server application using TCP sockets.

ALGORITHM :

Server:

1. Create a server socket
2. Wait for client to be connected.
3. Read Client's message and display it
4. Get a message from user and send it to client
5. Repeat steps 3-4 until the client sends "end"
6. Close all streams
7. Close the server and client socket
8. Stop

Client:

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

PROGRAM:

SERVER :

Program

```
import java.io.*;
import java.net.*;
classtcpchatserver
{
public static void main(String args[])throws Exception
{
PrintWritertoClient;
BufferedReaderfromUser, fromClient;
```

```
try
{
ServerSocketSrv = new ServerSocket(5555);
System.out.print("\nServer started\n");
Socket Clt = Srv.accept();
System.out.println("Client connected");
toClient = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(Clt.getOutputStream())), true);
fromClient = new BufferedReader(new
InputStreamReader(Clt.getInputStream()));
fromUser = new BufferedReader(new
InputStreamReader(System.in));
String CltMsg, SrvMsg;
while(true)
{
CltMsg= fromClient.readLine();
if(CltMsg.equals("end"))
break;
else
{
System.out.println("\nServer<<< " +
CltMsg);
System.out.print("Message to Client : ");
SrvMsg = fromUser.readLine();
toClient.println(SrvMsg);
}
}
System.out.println("\nClient Disconnected");
fromClient.close();
toClient.close();
fromUser.close();
Clt.close();
Srv.close();
}
catch (Exception E)
{
System.out.println(E.getMessage());
}
}
}
CLIENT:
import java.io.*;
import java.net.*;
```

```
classtcpchatclient
{
public static void main(String args[])throws Exception
{
Socket Clt;
PrintWritertoServer;
BufferedReaderfromUser, fromServer;
try
{
if (args.length> 1)
{
System.out.println("Usage: java hostipaddr");
System.exit(-1);
}
if (args.length == 0)
Clt = new Socket(InetAddress.getLocalHost(),5555);
else
Clt = new Socket(InetAddress.getByName(args[0]),
5555);
toServer = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(Clt.getOutputStream())), true);
fromServer = new BufferedReader(new
InputStreamReader(Clt.getInputStream()));
fromUser = new BufferedReader(new
InputStreamReader(System.in));
String CltMsg, SrvMsg;
System.out.println("Type \"end\" to Quit");
while (true)
{
System.out.print("\nMessage to Server : ");
CltMsg = fromUser.readLine();
toServer.println(CltMsg);
if (CltMsg.equals("end"))
break;
SrvMsg = fromServer.readLine();
System.out.println("Client <<< " + SrvMsg);
}
}
catch(Exception E)
{
System.out.println(E.getMessage());
}
}
```

}

OUTPUT:

Chat server

Server started
Client connected

Server<<< hi
Message to Client : hello

Server<<< welcome
Message to Client : thank you

Client Disconnected

Chat client

Type "end" to Quit

Message to Server : hi
Client <<< hello

Message to Server : welcome
Client <<< thank you

Message to Server : end

RESULT:

Thus chat experiment is performed using TCP socket programming.

EX NO : 5 IMPLEMENT - CONCURRENT SERVER USING TCP SOCKETS

AIM :

To write a java program to implement concurrent server using TCP sockets

THEORY:

Concurrent server is server which able to handle multiple request at the same time. Concurrent server create child server for each request. Concurrent server accept the request from client , this request pass to child server and concurrent server able to accept new request, where child server process that request and response to client.

Using Thread objects.

The simple server we have produced so far sits looping forever waiting to accept a client request. If several requests come 'at once' the server will deal with the first one and the others are queued while this is done. Having finished with the first client the next is dealt with. If any more client requests arrive these are queued up to the maximum specified in the listening queue. If the queue is full then any extra requests will be 'lost'. If the processing being done by the server takes a long time for each request then some clients may have to wait a long time. If we could have a server for each client request then each client would probably get a better response time! This may be achieved by the original server producing (**spawn**ing) a child process to deal with each accepted client connection.

The present server is a single thread of processing. The *Thread class* allows us to create subthreads form this main one. As ever, **Java** lets us do this in more than one way – by extending the *Thread class* or by implementing the *Runnable interface*! We shall use the technique of extending the Thread class.

ALGORITHM:

SERVER:

1. Start the program.
2. Initialize the port number.
3. Client sends request to the server.
4. If the connection is established.
5. Stop the program.

CLIENT:

1. Start the program.
2. Initialize the port number.
3. The server will process the client request.

4. The server will process the client request.
5. Initialize the valid address and get the system name.
6. Print the message is received.
7. Stop the program.

PROGRAM:

SERVER:

```
import java.io.*;
import java.net.*;
classconcurrentserver
{
public static void main(String args[]) throws Exception
{
int count=0;
Thread t=Thread.currentThread();
ServerSocketss=new ServerSocket(555);
while(true)
{
try
{
Socket S=ss.accept();
count ++;
conserver c=new conserver(S,count);
}
catch(Exception e){}
}
}
}
class conserver implements Runnable
{
Thread t;
Socket S;
int c;
conserver(Socket S1,int count)
{
t=new Thread(this,"Demo");
t.start();
S=S1;
c=count;
```

```
}
public void run()
{
try
{
DataInputStreaminp=new DataInputStream(S.getInputStream());
DataOutputStream out=new DataOutputStream(S.getOutputStream());
String sentence="Enter the string";
String newsentence;
out.writeBytes(sentence+"\n");
newsentence=inp.readLine();
System.out.println("From client"+c+":"+newsentence);
out.writeBytes(newsentence+"\n");
}
catch(Exception e){}
}
}
```

CLIENT:

```
import java.net.*;
import java.io.*;
importjava.lang.String;
classconcurrentclient
{
public static void main(String args[]) throws Exception
{
String sentence,newsentence,newsentence1;
DataInputStream in=new DataInputStream(System.in);
Socket cs=new Socket("LOCALHOST",555);
DataInputStreaminp=new DataInputStream(cs.getInputStream());
DataOutputStream out=new DataOutputStream(cs.getOutputStream());
sentence=inp.readLine();
System.out.println(sentence);
newsentence=in.readLine();
out.writeBytes(newsentence+'\n');
newsentence1=inp.readLine();
System.out.println("From Server:"+newsentence1);
cs.close();
}
}
```

OUTPUT:

JAVA CLIENT1
Enter the string
Hi
From Server:Hi
JAVA CLIENT2
Enter the string
Hello
From Server:Hello
JAVA CLIENT3
Enter the string
Welcome to networks lab
From Server:Welcome to networks lab
JAVA CLIENT4
Enter the string
bye
From Server:bye
JAVA SERVER
From client1:Hi
From client2:Hello
From client3:WelcoMe to networks lab
From client4:bye

RESULT:

Thus the program for concurrent server has been completed and output has been verified
successfully.

EX NO : 6 IMPLEMENT - PING COMMAND USING RAW SOCKETS

AIM:

To write a java program to implement ping command in raw sockets.

THEORY:

Syntax

ping [**-t**] [**-a**] [**-n** *Count*] [**-l** *Size*] [**-f**] [**-i** *TTL*] [**-v** *TOS*] [**-r** *Count*] [**-s** *Count*] [{**-j** *HostList* | **-k** *HostList*}] [**-w** *Timeout*] [*TargetName*]

Parameters

-t : Specifies that ping continue sending Echo Request messages to the destination until interrupted. To interrupt and display statistics, press CTRL-BREAK. To interrupt and quit ping, press CTRL-C.

-a : Specifies that reverse name resolution is performed on the destination IP address. If this is successful, ping displays the corresponding host name.

-n *Count* : Specifies the number of Echo Request messages sent. The default is 4.

-l *Size* : Specifies the length, in bytes, of the Data field in the Echo Request messages sent. The default is 32. The maximum *size* is 65,527.

-f : Specifies that Echo Request messages are sent with the Don't Fragment flag in the IP header set to 1. The Echo Request message cannot be fragmented by routers in the path to the destination. This parameter is useful for troubleshooting path Maximum Transmission Unit (PMTU) problems.

-i *TTL* : Specifies the value of the TTL field in the IP header for Echo Request messages sent. The default is the default TTL value for the host. For Windows XP hosts, this is typically 128. The maximum *TTL* is 255.

-v *TOS* : Specifies the value of the Type of Service (TOS) field in the IP header for Echo Request messages sent. The default is 0. *TOS* is specified as a decimal value from 0 to 255.

-r *Count* : Specifies that the Record Route option in the IP header is used to record the path taken by the Echo Request message and corresponding Echo Reply

message. Each hop in the path uses an entry in the Record Route option. If possible, specify a *Count* that is equal to or greater than the number of hops between the source and destination. The *Count* must be a minimum of 1 and a maximum of 9.

-s *Count* : Specifies that the Internet Timestamp option in the IP header is used to record the time of arrival for the Echo Request message and corresponding Echo Reply message for each hop. The *Count* must be a minimum of 1 and a maximum of 4.

-j *HostList* : Specifies that the Echo Request messages use the Loose Source Route option in the IP header with the set of intermediate destinations specified in *HostList*. With loose source routing, successive intermediate destinations can be separated by one or multiple routers. The maximum number of addresses or names in the host list is 9. The host list is a series of IP addresses (in dotted decimal notation) separated by spaces.

-k *HostList* : Specifies that the Echo Request messages use the Strict Source Route option in the IP header with the set of intermediate destinations specified in *HostList*. With strict source routing, the next intermediate destination must be directly reachable (it must be a neighbor on an interface of the router). The maximum number of addresses or names in the host list is 9. The host list is a series of IP addresses (in dotted decimal notation) separated by spaces.

-w *Timeout* : Specifies the amount of time, in milliseconds, to wait for the Echo Reply message that corresponds to a given Echo Request message to be received. If the Echo Reply message is not received within the time-out, the "Request timed out" error message is displayed. The default time-out is 4000 (4 seconds).

***TargetName* :** Specifies the destination, which is identified either by IP address or host name.

ALGORITHM:

1. Create a RAW socket in the client program.
2. Get the name of the host whose IP address is to resolve using ICMP.
3. Pass this name to the ICMP server through this socket.
4. The server will respond with the IP address of the host.
5. Receive the response and print it.

PROGRAM:

PING SERVER:

```
import java.io.*;
import java.net.*;
importjava.util.*;
classPingServer
{
public static void main(String[] args)
{
try
{
ServerSocketss=new ServerSocket(2156);
Socket s=ss.accept();
if(s.isConnected())
System.out.println("Connected ...");
System.out.println("Listening ...");
DataInputStream dis=new DataInputStream(s.getInputStream());
DataOutputStream dos=new DataOutputStream(s.getOutputStream());
int no=0;
String ip="";
if((dis.readUTF()).equals("P"))
{
System.out.println("Getting No. Of Packets ...");
no=dis.readInt();
}
if((dis.readUTF()).equals("A"))
{
System.out.println("Getting the Address ...");
ip=dis.readUTF();
}
Process p=Runtime.getRuntime().exec("ping -c "+no+" "+ip);
System.out.println("Running  ping -c "+no+" "+ip);
BufferedReaderbr=new BufferedReader(new InputStreamReader(p.getInputStream()));
String ipline=br.readLine();
while(ipline != null )
{
dos.writeUTF(ipline);
ipline=br.readLine();
}
dis.close();
dos.close();
}catch(Exception x)
{
```

```
x.printStackTrace();
}
}
}
```

PING CLIENT :

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

public class PingClient
{
public static void main(String[] args)
{
try
{
Socket s=new Socket("localhost",2156);
BufferedReaderbr=new BufferedReader(new InputStreamReader(System.in));
if(s.isConnected())
System.out.println("Connected !!");
Scanner in=new Scanner(System.in);
DataInputStream is=new DataInputStream(s.getInputStream());
DataOutputStreamos=new DataOutputStream(s.getOutputStream());
System.out.println("How many Packets You want to send ? ");
int no=in.nextInt();
System.out.println("Address to be pinged :");
String ip=br.readLine();
os.writeUTF("P");
os.writeInt(no);
os.writeUTF("A");
os.writeUTF(ip);
String pingline=is.readUTF();
while(pingline != null )
{
System.out.println(pingline);
pingline=is.readUTF();
}
os.flush();
os.close();
is.close();
}catch(Exception x)
{
```

```
}
}
}
```

OUTPUT:

Pingserver

```
Connected ...
Listening ...
Getting No. Of Packets ...
Getting the Address ...
Running  ping -n 5 192.168.100.1
```

pingclient

```
Connected !!
How many Packets You want to send ?
5
Address to be pinged :
192.168.100.1

Pinging 192.168.100.1 with 32 bytes of data:
Reply from 192.168.100.1: bytes=32 time<1ms TTL=128
Reply from 192.168.100.1: bytes=32 time<1ms TTL=128
Reply from 192.168.100.1: bytes=32 time<1ms TTL=128
Reply from 192.168.100.1: bytes=32 time<1ms TTL=128
Reply from 192.168.100.1: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.100.1:
   Packets: Sent = 5, Received = 5, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
   Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

RESULT:

Thus the program for ping has been written in java and executed successfully.

EX NO :7 IMPLEMENT - TRACEROUTE COMMAND USING RAW SOCKETS

AIM:

To write a java program to implement trace route application using raw rockets.

THEORY:

The TRACERT diagnostic utility determines the route taken to a destination by sending Internet Control Message Protocol (ICMP) echo packets with varying IP Time-To-Live (TTL) values to the destination. Each router along the path is required to decrement the TTL on a packet by at least 1 before forwarding it, so the TTL is effectively a hop count. When the TTL on a packet reaches 0, the router should send an ICMP Time Exceeded message back to the source computer.

TRACERT determines the route by sending the first echo packet with a TTL of 1 and incrementing the TTL by 1 on each subsequent transmission until the target responds or the maximum TTL is reached. The route is determined by examining the ICMP Time Exceeded messages sent back by intermediate routers. Note that some routers silently drop packets with expired TTLs and are invisible to TRACERT.

TRACERT prints out an ordered list of the routers in the path that returned the ICMP Time Exceeded message. If the -d switch is used (telling TRACERT not to perform a DNS lookup on each IP address), the IP address of the near- side interface of the routers is reported.

ALGORITHM:

1. Create a RAW socket in the client program.
2. Get the name or ip address of the host for which the route is to be traced.
3. Pass the ip address to the server.
4. The server sends the icmp packet to destination.
5. The route is traced and printed at the client.
6. Close the connection between the server and client.

PROGRAM:

SERVER:

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

```
{
public static void main(String[] args)
{
try
{
ServerSocketss=new ServerSocket(2156);
System.out.println("Listening ...");
Socket s=ss.accept();
if(s.isConnected())
System.out.println("Connected ...");
DataInputStream dis=new DataInputStream(s.getInputStream());
DataOutputStream dos=new DataOutputStream(s.getOutputStream());
int no=0;
String ip="";
System.out.println("Getting the Address ...");
ip=dis.readUTF();
Process p=Runtime.getRuntime().exec("tracert " +ip);
System.out.println("Running  tracert "+ip);
BufferedReaderbr=new BufferedReader(new InputStreamReader(p.getInputStream()));
String traceline=br.readLine();
while(traceline != null )
{
dos.writeUTF(traceline);
traceline=br.readLine();
}
dis.close();
dos.close();
}catch(Exception x)
{
x.printStackTrace();
}
}
}
```

CLIENT :

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

public class TraceClient
{
public static void main(String[] args)
```

```
{
try
{
Socket s=new Socket("localhost",2157);
BufferedReaderbr=new BufferedReader(new InputStreamReader(System.in));
if(s.isConnected())
System.out.println("Connected !!");
Scanner in=new Scanner(System.in);
DataInputStream is=new DataInputStream(s.getInputStream());
DataOutputStreamos=new DataOutputStream(s.getOutputStream());
System.out.println("Address :");
String ip=br.readLine();
os.writeUTF(ip);
String traceline;
traceline=is.readUTF();
while(traceline != null)
{
System.out.println(traceline);
traceline=is.readUTF();
}
os.flush();
os.close();
is.close();
}catch(Exception x)
{
//x.printStackTrace();
}
}
}
```

OUTPUT:

Trace server:

Listening ...
Connected ...
Getting the Address ...
Running tracert www.yahoo.com

Trace client:

Connected !!
Address :
www.yahoo.com

Tracing route to ds-sg-fp3.wg1.b.yahoo.com [106.10.139.246]

over a maximum of 30 hops:

```
  1    1 ms     1 ms     1 ms  192.168.1.1
  2   31 ms    24 ms    26 ms  117.217.192.1
  3   24 ms    26 ms    39 ms  218.248.161.234
  4  146 ms   143 ms   142 ms  218.248.255.70
  5    *      209 ms     *     115.114.130.49.STATIC-Chennai.vsnl.net.in
[115.114.130.49]
  6   92 ms    91 ms     *     ix-4-2.tcore1.CXR-Chennai.as6453.net [180.87.36.9]
  7  122 ms   123 ms   156 ms  if-3-3.tcore2.CXR-Chennai.as6453.net
[180.87.36.6]
  8  130 ms   122 ms   129 ms  if-6-2.tcore2.SVW-Singapore.as6453.net
[180.87.37.14]
  9  122 ms   129 ms   120 ms  180.87.15.122
 10  122 ms   123 ms   124 ms  ae-5.msr1.sg3.yahoo.com [203.84.209.87]
 11  124 ms   120 ms   123 ms  ae-2.clr1-a-gdc.sg3.yahoo.com [106.10.128.3]
 12  137 ms   124 ms   124 ms  et-17-1.fab3-1-gdc.sg3.yahoo.com [106.10.128.17]
 13  402 ms   397 ms   402 ms  po-11.bas2-1-prd.sg3.yahoo.com [106.10.128.71]
 14  126 ms   130 ms   122 ms  ir1.fp.vip.sg3.yahoo.com [106.10.139.246]
```

Trace complete.

RESULT:

Thus trace route application is implemented and verified.

EX NO: 8 IMPLEMENT - REMOTE COMMAND EXECUTION USING TCP SOCKETS

AIM:

To write a java program to implement remote command execution using TCP sockets.

ALGORITHM:

SERVER:

1.Create server and client socket.
2.Create a runtime object using getRuntime() method.
3.Create a process object p.
4.Read the command entered by the user at the client using input streams.
5.Execute the command using the exec().

CLIENT:

1.Create a client socket corresponding to the required server and port.
2.Promote the user to enter the command.
3.Read the command using input streams.
4.Write the command to the server using output streams.
5.Lose the streams and socket.

PROGRAM:

RemoteServer:

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

class RemoteServer
{
public static void main(String args[])
{
try
{
int Port;
BufferedReaderBuf =new BufferedReader(new
InputStreamReader(System.in));
System.out.print(" Enter the Port Address : " );
Port=Integer.parseInt(Buf.readLine());
ServerSocketss=new ServerSocket(Port);
```

```
System.out.println(" Server is Ready To Receive a Command. ");
System.out.println(" Waiting ..... ");
Socket s=ss.accept();
if(s.isConnected()==true)
        System.out.println(" Client Socket is Connected Succecfully. ");
InputStream in=s.getInputStream();
OutputStreamou=s.getOutputStream();
BufferedReaderbuf=new BufferedReader(new
InputStreamReader(in));
String cmd=buf.readLine();
PrintWriterpr=new PrintWriter(ou);
pr.println(cmd);
Runtime H=Runtime.getRuntime();
Process P=H.exec(cmd);
System.out.println(" The " + cmd + " Command is Executed Successfully. ");
pr.flush();
pr.close();
ou.close();
in.close();
}
catch(Exception e)
{
System.out.println(" Error : " + e.getMessage());
}
}
}
```

OUTPUT:

RESULT:

Thus remote command execution using TCP sockets is implemented in java.

EX NO: 9 IMPLEMENT - DNS USING UDP SOCKETS

AIM:

To implement DNS using UDP sockets in java.

THEORY:

Short for **D***omain* **N***ame* **S***ystem* (or **S***ervice* or **S***erver*), an Internet service that translates *domain names* into IP addresses. Because domain names are alphabetic, they're easier to remember. The Internet however, is really based on IP addresses. Every time you use a domain name, therefore, a DNS service must translate the name into the corresponding IP address. For example, the domain name *www.example.com* might translate to *198.105.232.4*.

The Domain Name System distributes the responsibility of assigning domain names and mapping those names to IP addresses by designating authoritative name servers for each domain. Authoritative name servers are assigned to be responsible for their supported domains, and may delegate authority over subdomains to other name servers. This mechanism provides distributed and fault tolerant service and was designed to avoid the need for a single central database.The DNS system is, in fact, its own network. If one DNS server doesn't know how to translate a particular domain name, it asks another one, and so on, until the correct IP address is returned.

ALGORITHM:

1. Start the program
2. Enter the system name (dn) after the connection with the server is established.
3. Call the subroutine resolver and pass the system name to the resolver.
4. Open the host files in the DNS server.
5. Check the system name with the name stored in the host file until the end of the file.
6. If the name matches, then fetch the corresponding IP address and display the IP address. Go to step8.
7. If the match is not found, then display the message as system is not logged on.
8. Stop the process.

PROGRAM

DNS CLIENT

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

```
importjava.util.*;
classdclient
{
public static void main(String args[])
{
try
{
DatagramSocket client=new DatagramSocket();
InetAddressaddr=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 DOMAIN NAME OR IP address");
String str=in.readLine();
sendbyte=str.getBytes();
DatagramPacket sender=new DatagramPacket(sendbyte,sendbyte.length,addr,1309);
client.send(sender);
DatagramPacket receiver= new DatagramPacket(receivebyte,receivebyte.length);
client.receive(receiver);
String s=new String(receiver.getData());
System.out.println("IP adddress or DOMAIN NAME :"+s.trim());
client.close();
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

DNS SERVER

```
import java.io.*;
import java.net.*;
import java.util.*;
class dserver
{
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);
InetAddressaddr=receiver.getAddress();
int port=receiver.getPort();
String ip[]={"165.165.80.80","165.165.79.1"};
String name[]={"www.aptitude source.com","www.Sharifguys.com"};
for(int i=0;i<ip.length;i++)
{
if(s.equals(ip[i]))
{
sendbyte=name[i].getBytes();
DatagramPacket sender=new DatagramPacket(sendbyte,sendbyte.length,addr, port);
server.send(sender);
break;
}
else if(s.equals(name[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:

Java Server

165.165.80.80

165.165.79.1

Java Client

Enter domain name or IP address
165.165.80.80
Ip address or domain name: www.aptitude source.com
165.165.79.1
Ip address or domain name: www.Sharifguys.com

RESULT:

Thus the java program to implement Domain Name System is successfully executer and verified.

EX NO :10 IMPLEMENT - SLIDING WINDOW PROTOCOL USING SOCKET PROGRAMMING

AIM:

To write a java code for implementing sliding window protocol using socket programming.

ALGORITHM:

SERVER:

7. Start the program
8. Get the window size from the client
9. Receive the data from the client the size of the window.
10. Send the acknowledgement to the client, if data is received correctly.
11. Send negative acknowledgement to the client, if data is not received.

CLIENT:

1. Start the program
2. Get the data to be sent to the server.
3. Get the window size from the user.
4. Send the data of the size of the window to the server.
5. If negative acknowledgement is received resend the data to the server.
6. After sending all the data terminate the connection with the server.

PROGRAM:

SERVER:

```
import java.io.*;
import java.net.*;
importjava.util.*;
public class SlidingServer
{
public static void main(String args[]) throws Exception
                {
ServerSocketss=new ServerSocket(1234);
String msg,ack;
Scanner in=new Scanner(System.in);
Socket s=ss.accept();
BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));
BufferedReader read=new BufferedReader(new InputStreamReader(s.getInputStream()));
PrintWriter write=new PrintWriter(s.getOutputStream(),true);
System.out.println("SLIDING WINDOW PROTOCOL");
int k=0,c=0,count=0;
msg=read.readLine();
```

```
intlen=msg.length();
System.out.println("length is "+len);
while(!msg.trim().equals("exit"))
{
if(msg!=null)
System.out.println("Data Received : "+msg);
for(int l=0;l<len;l++)
                {
System.out.println("\nAcknowledgement for segment              "+k+"[yes]/[no]");
ack=bf.readLine();
if(!ack.trim().equals("yes"))
{
count++;
if(count==1)
c=k;
  }
write.println(ack);
k++;
    }
if(c!=0)
      k=c;
msg=read.readLine();
len=msg.length();
count=0;
c=0;
}
s.close();
ss.close();
read.close();
write.close();
}
}
```

CLIENT:

```
import java.io.*;
import java.net.*;
importjava.util.*;
public class SlidingClient{
public static void main(String args[]) throws Exception{
try{
Socket s=new Socket("localhost",1234);
Scanner in=new Scanner(System.in);
String msg,msg1,ack;
```

```
intws,ml,ind=-1;
BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));
BufferedReader read=new BufferedReader(new InputStreamReader(s.getInputStream()));
PrintWriter write=new PrintWriter(s.getOutputStream(),true);
BufferedReader scan=new BufferedReader(new InputStreamReader(System.in));
System.out.println("SLIDING WINDOW PROTOCOL");
System.out.println("Data to Send :");
msg=scan.readLine();
ml=msg.length();
intarr[]=new int[ml];
System.out.println("Messge length "+ml);
System.out.println("Window size");
ws=in.nextInt();
if(msg!=null)
System.out.println(msg);
int k=0;
int c=0,count=0;
do
{
      msg1=msg.substring(k,k+ws);
if(msg1!=null)
      {
System.out.println(msg1);
write.println(msg1);
for(int l=0;l<msg1.length();l++)
      {
ack=read.readLine();
System.out.println(ack);
if(!ack.trim().equals("no"))
          {
c++;
           }
else {
if(count==0)
ind=c;
count++;
c++;
           }
       }
if(ind==-1)
   k=k+ws;
else
{
```

```
    k=ind;
    c=ind;
}
if((ml-k)<ws)
ws=ml-k;
count=0;
ind=-1;
}
}
while(k<ml);
write.println("exit");
s.close();
read.close();
write.close();
scan.close();
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

OUTPUT:

```
SERVER:
SLIDING WINDOW PROTOCOL
length is 4
Data Received : Hell

Acknowledgement for segment          0[yes]/[no]
1
Acknowledgement for segment          1[yes]/[no]
3
Acknowledgement for segment          2[yes]/[no]
4
Acknowledgement for segment          3[yes]/[no]
5
Data Received : o

Acknowledgement for segment          4[yes]/[no]
```

1

CLIE NT:
SLIDING WINDOW PROTOCOL
Data to Send :
Hello
Messge length 5
Window size
4
Hello
Hell
1
3
4
5
o
1

RESULT:
Thus sliding window protocol is executed in java.

EX. 11 IMPLEMENT – CRC ERROR DETECTION CODE FOR GIVEN MESSAGE

AIM:

To write a java program to implement CRC to detect errors for a given message.

THEORY:

To compute an *n*-bit binary CRC, line the bits representing the input in a row, and position the (*n* + 1)-bit pattern representing the CRC's divisor (called a "polynomial") underneath the left-hand end of the row.

Start with the message to be encoded:

```
11010011101100
```

This is first padded with zeroes corresponding to the bit length *n* of the CRC. Here is the first calculation for computing a 3-bit CRC:

```
11010011101100 000 <--- input right padded by 3 bits
1011               <--- divisor (4 bits) = x³+x+1
------------------
01100011101100 000 <--- result
```

If the input bit above the leftmost divisor bit is 0, do nothing. If the input bit above the leftmost divisor bit is 1, the divisor is XORed into the input (in other words, the input bit above each 1-bit in the divisor is toggled). The divisor is then shifted one bit to the right, and the process is repeated until the divisor reaches the right-hand end of the input row. Here is the entire calculation:

```
11010011101100 000 <--- input right padded by 3 bits
1011               <--- divisor
01100011101100 000 <--- result
 1011              <--- divisor ...
00111011101100 000
  1011
00010111101100 000
   1011
00000001101100 000
       1011
00000000110100 000
        1011
00000000011000 000
         1011
00000000001110 000
          1011
00000000000101 000
           101 1
-----------------
00000000000000 100 <--- remainder (3 bits)
```

Since the leftmost divisor bit zeroed every input bit it touched, when this process ends the only bits in the input row that can be nonzero are the n bits at the right-hand end of the row. These *n*

bits are the remainder of the division step, and will also be the value of the CRC function (unless the chosen CRC specification calls for some postprocessing).

The validity of a received message can easily be verified by performing the above calculation again, this time with the check value added instead of zeroes. The remainder should equal zero if there are no detectable errors.

```
11010011101100 100 <--- input with check value
1011               <--- divisor
01100011101100 100 <--- result
 1011              <--- divisor ...
00111011101100 100
......
00000000001110 100
          1011
00000000000101 100
           101 1
------------------
                 0 <--- remainder
```

PROGRAM:

```
import java.io.*;
class crc_gen
{
 public static void main(String args[]) throws IOException
{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
int[] data;
int[] div;
int[] divisor;
int[] rem;
int[] crc;
int data_bits, divisor_bits, tot_length;
System.out.println("Enter number of data bits : ");
data_bits=Integer.parseInt(br.readLine());
data=new int[data_bits];
System.out.println("Enter data bits : ");
for(int i=0; i<data_bits; i++)
data[i]=Integer.parseInt(br.readLine());
System.out.println("Enter number of bits in divisor : ");
divisor_bits=Integer.parseInt(br.readLine());
divisor=new int[divisor_bits];
System.out.println("Enter Divisor bits : ");
for(int i=0; i<divisor_bits; i++)
divisor[i]=Integer.parseInt(br.readLine());
System.out.print("Data bits are : ");
```

```
for(int i=0; i< data_bits; i++)
System.out.print(data[i]);
System.out.println();

System.out.print("divisor bits are : ");
for(int i=0; i< divisor_bits; i++)
System.out.print(divisor[i]);
System.out.println();
tot_length=data_bits+divisor_bits-1;
div=new int[tot_length];
rem=new int[tot_length];
crc=new int[tot_length];
   /*------------------ CRC GENERATION----------------------*/
      for(int i=0;i<data.length;i++)
         div[i]=data[i];
System.out.print("Dividend (after appending 0's) are : ");
for(int i=0; i< div.length; i++)
System.out.print(div[i]);
System.out.println();
for(int j=0; j<div.length; j++){
rem[j] = div[j];
}
rem=divide(div, divisor, rem);
for(int i=0;i<div.length;i++)          //append dividend and ramainder
{
         crc[i]=(div[i]^rem[i]);
}
System.out.println();
System.out.println("CRC code : ");
for(int i=0;i<crc.length;i++)
System.out.print(crc[i]);

   /*--------------------ERROR DETECTION---------------------*/
System.out.println();
System.out.println("Enter CRC code of "+tot_length+" bits : ");
for(int i=0; i<crc.length; i++)
crc[i]=Integer.parseInt(br.readLine());
System.out.print("crc bits are : ");
for(int i=0; i< crc.length; i++)
System.out.print(crc[i]);
System.out.println();
for(int j=0; j<crc.length; j++){
            rem[j] = crc[j];
```

```
 }
rem=divide(crc, divisor, rem);
for(int i=0; i< rem.length; i++)
{
 if(rem[i]!=0)
{
System.out.println("Error");
break;
}
if(i==rem.length-1)
System.out.println("No Error");
}
System.out.println("THANK YOU.... :)");
}
static int[] divide(int div[],int divisor[], int rem[])
{
int cur=0;
while(true)
{
for(int i=0;i<divisor.length;i++)
rem[cur+i]=(rem[cur+i]^divisor[i]);
while(rem[cur]==0 && cur!=rem.length-1)
cur++;
if((rem.length-cur)<divisor.length)
 break;
}
return rem;
}
}
```

OUTPUT :

```
Enter number of data bits :
7
Enter data bits :
1
0
1
1
0
0
```

1
Enter number of bits in divisor :
3
Enter Divisor bits :
1
0
1
Data bits are : 1011001
divisor bits are : 101
Dividend (after appending 0's) are : 101100100

CRC code :
101100111
Enter CRC code of 9 bits :
1
0
1
1
0
0
1
0
1
crc bits are : 101100101
Error
Press any key to continue...

RESULT:
Thus the program to CRC algorithm is written and successfully executed.

EX. NO. 12 IMPLEMENT – SNMP PROTOCOL USING SOCKET PROGRAMMING

AIM:

To write a java program to implement SNMP protocol.

THEORY:

The program listed below illustrates the use of the communication interface class to open a connection to a remote APC UPS device. The program has two important functions named as **snmpSet** and **snmpGet.** These methods can be used as standalone methods.
In main method

- snmpSet is called to turn **ON,** UPS *outlet group1* by setting Value=1
- snmpGet is called to get the ***basic state*** of UPS device.

PROGRAM:

```
import java.net.*;
import snmp.*; //snmp.jar file
public class SNMPSampleSetGet
{
public static final String READ_COMMUNITY = "public";
public static final String WRITE_COMMUNITY= "private";
public static final int mSNMPVersion =0; // 0 represents SNMP version=1
public static final String OID_OUTLET_GROUP1="1.3.6.1.4.1.318.1.1.1.12.3.2.1.3.1";
public static final String OID_UPS_BASIC_STATE= "1.3.6.1.4.1.318.1.1.1.11.1.1.0";
public static void main(String args[])
{
boolean m_bOnBattery = false;
boolean m_bOnLine = false ;
boolean m_bReplaceBattery = false ;
String strIPAddress = "172.20.1.150";
/*
* OID_UPS_BASIC_STATE can be used to get (APC UPS) Basic State
* see http://osdir.com/ml/network.bb4.general/2003-11/msg00302.html
*/
SNMPSampleSetGet objSNMP = new SNMPSampleSetGet();
////////////////////////////////////////////////
//Get Basic state of UPS
////////////////////////////////////////////////
String strState =objSNMP.snmpget(strIPAddress, READ_COMMUNITY,
mSNMPVersion, OID_UPS_BASIC_STATE);
```

```
// The snmpget returns strstate =
//00010100000000000010000000000000000000000000000000000000000000000
if(strstate!=null&&strstate.length()!=0)
{
m_bOnBattery = (strstate.charAt(1) == '0') ? false : true;
m_bOnLine = (strstate.charAt(3) == '0') ? false : true;
m_bReplaceBattery = (strstate.charAt(4) == '0') ? false : true;
}
////////////////////////////////////////////////////////////////////////
//Set Value=1 to trun ON UPS OUTLET Group1
//Value=2 to trun OFF UPS OUTLET Group1
////////////////////////////////////////////////////////////////////////
int Value = 1;
objSNMP.snmpset(strIPAddress, WRITE_COMMUNITY, mSNMPVersion,
OID_UPS_OUTLET_GROUP1,Value);
}
/**
* This method set the Value on the device
* community=WRITE_COMMUNITY
*/
public void snmpSet(String strIPAddress, String community, int iSNMPVersion,
String strOID, int Value)
{
try
{
InetAddress hostAddress = InetAddress.getByName(strIPAddress);
SNMPv1CommunicationInterface comInterface = New
SNMPv1CommunicationInterface(iSNMPVersion,
hostAddress, community);
SNMPVarBindList newVars = new SNMPVarBindList();
System.out.println("Setting value corresponding to OID " + strOID);
newVars = comInterface.setMIBEntry(strOID, new SNMPInteger(Value));
String valueString = new String();
for (int i = 0; i < newVars.size(); ++i)
{
valueString += newVars.toString()+ "\n";
}
System.out.println("valueString: " + valueString);
}
catch(Exception e)
{
System.out.println("Exception during SNMP Set operation: " + e + "\n");
}
```

```
}
/*
* This method process the request and Get the Value on the device
* @returns String. community=READ_COMMUNITY
*/
public String snmpGet(String strIPAddress, String community, int iSNMPVersion, String
strOID)
{
String str="";
try
{
InetAddress hostAddress = InetAddress.getByName(strIPAddress);
System.out.println("hostAddress =" + hostAddress);
System.out.println("community =" + community);
SNMPv1CommunicationInterface comInterface =
new SNMPv1CommunicationInterface(iSNMPVersion, hostAddress,
community);
System.out.println("strOID =" + strOID);
// returned Vector has just one pair inside it.
SNMPVarBindList newVars = comInterface.getMIBEntry(strOID);
// extract the (OID,value) pair from the SNMPVarBindList;
//the pair is just a two-element
// SNMPSequence
SNMPSequence pair = (SNMPSequence)(newVars.getSNMPObjectAt(0));
// extract the object identifier from the pair; it's the first element in the sequence
SNMPObjectIdentifier snmpOID =
(SNMPObjectIdentifier)pair.getSNMPObjectAt(0);
//extract the corresponding value from the pair;
// it's the second element in the sequence
SNMPObject snmpValue = pair.getSNMPObjectAt(1);
str = snmpValue.toString();
}
catch(Exception e)
{
System.out.println("Exception during SNMP operation: " + e + "\n");
}
System.out.println("Retrieved value: "+ str);
return str;
}
}
```

CYCLE -2
NETWORK SIMULATOR 2

EX.NO:1 Generating network topology using NS2

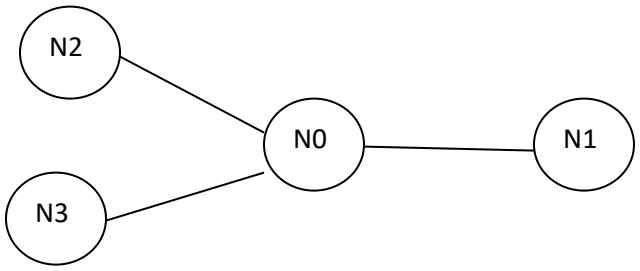
AIM:
To write a tcl script to create a simple topology in NS2.

EXPLANATION:
A simple topology of four nodes are created. A duplex link is established between the node n0 and n1, n0 and n2 and n0 and n3.

PROGRAM:

```
set ns [new Simulator]
#create file for analysis mode
set tr [open out.tr w]
$ns trace-all $tr
#create file for Animation Mode
set namtr [open out.nam w]
$ns namtrace-all $namtr
#Create Node
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
#Create Link
$ns duplex-link $n0 $n1 10Mb 5ms DropTail
$ns duplex-link $n2 $n0 10Mb 5ms DropTail
$ns duplex-link $n3 $n0 10mb 5ms DropTail
#Create Orientation
$ns duplex-link-op $n0 $n1 orient right
$ns duplex-link-op $n0 $n2 orient left-up
$ns duplex-link-op $n0 $n3 orient left-down
$ns at 10.0 "$ns halt"
$ns run
```

OUTPUT:



EX.2 CREATE UDP DATA TRAFFIC

AIM:
To create a UDP traffic between two nodes in a given scenario.

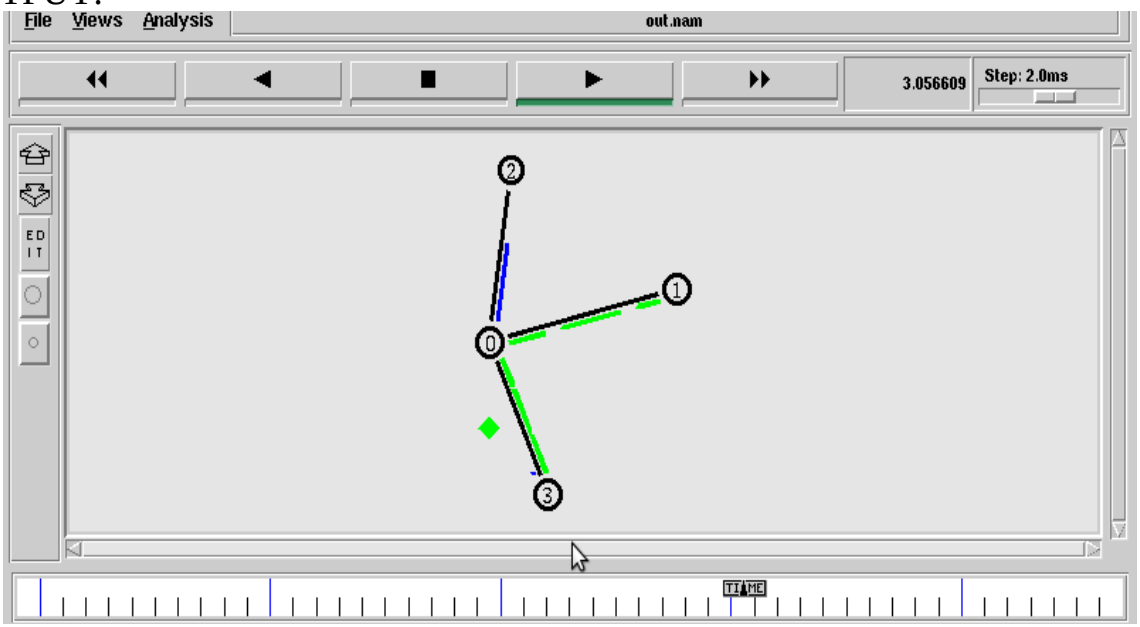
EXPLANATION:
A simple topology of four nodes is created. A duplex link is established between the node n0 and n2, n1 and n2 and n2 and n3. UDP CBR traffic is created between the nodes n1 and n3 and another CBR traffic between n2 and n3.

PROGRAM:

```
set ns [new Simulator]
set nf [open out.nam w]
$ns namtrace-all $nf
proc finish {} {
        global ns nf
        $ns flush-trace
        close $nf
        exec nam out.nam &
        exit 0
}

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n3 $n2 1Mb 10ms DropTail

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

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

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

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

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

set null0 [new Agent/Null]
$ns attach-agent $n3 $null0

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

$ns at 0.5 "$cbr0 start"
$ns at 1.0 "$cbr1 start"
$ns at 4.0 "$cbr1 stop"
$ns at 4.5 "$cbr0 stop"

$udp0 set class_ 1
$udp1 set class_ 2

$ns color 1 Blue
$ns color 2 Red

$ns duplex-link-op $n2 $n3 queuePos 0.5
$ns duplex-link $n3 $n2 1Mb 10ms SFQ


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

OUTPUT:



EX.3 TCP DATA TRAFFIC

AIM:
To create tcp data traffic between two nodes in a given scenario.

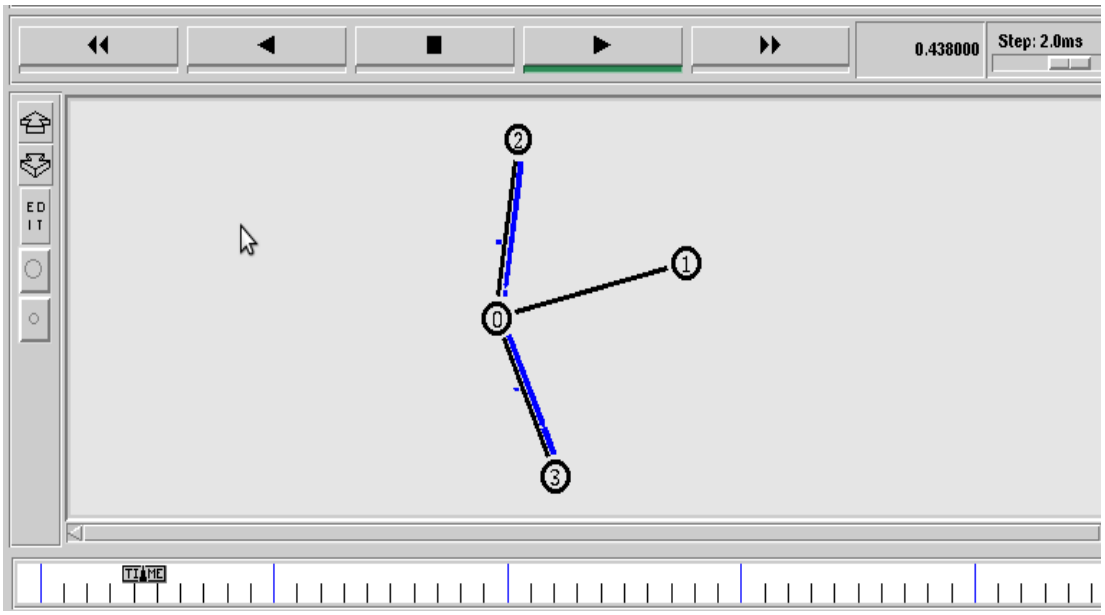
EXPLANATION:
A simple topology of four nodes is created. A duplex link is established between the node n0 and n2, n1 and n2 and n2 and n3. TCP FTP traffic is created between the nodes n1 and n3 and another TCP traffic between n2 and n3.

PROGRAM:

```
set ns [new Simulator]
set nf [open out.nam w]
$ns namtrace-all $nf
proc finish {} {
        global ns nf
        $ns flush-trace
        close $nf
        exec nam out.nam &
        exit 0
}
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n0 $n3 1Mb 10ms DropTail

#create a tcp connection and attach to node n1
set tcp1 [new Agent/TCP]
$ns attach-agent $n1 $tcp1
$tcp1 set window_ 8
$tcp1 set fid_ 1

#create a tcp connection and attach to node n2
set tcp2 [new Agent/TCP]
$ns attach-agent $n2 $tcp2
$tcp2 set window_ 8
$tcp2 set fid_ 2

#create the sink nodes 1 and 2
set sink1 [new Agent/TCPSink]
set sink2 [new Agent/TCPSink]

#attach sink 1 and 2 to node n3
$ns attach-agent $n3 $sink1
```

```
$ns attach-agent $n3 $sink2

$ns connect $tcp1 $sink1
$ns connect $tcp2 $sink2

#Create FTP applications and attach them to agents
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2

$ns at 0.1 "$ftp1 start"
$ns at 0.5 "$ftp2 start"
$ns at 3.5 "$ftp1 stop"
$ns at 5.0 "$ftp2 stop"

$ftp1 set class_ 1
$ftp2 set class_ 2

$ns color 1 Red
$ns color 2 Blue

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

OUTPUT:



EX.3 WIRELESS SCENARIO

AIM:

To create a simple wireless scenario in NS2

EXPLANATION:

A simple topology of three nodes is created. UDP CBR traffic is created between the nodes n1 and n0 and another CBR traffic between n2 and n1.

PROGRAM:

```
# Define options
#
# ======================================================================
set val(chan)   Channel/WirelessChannel     ;# channel type
set val(prop)         Propagation/TwoRayGround    ;# radio-propagation
model
set val(netif)        Phy/WirelessPhy                   ;# network
interface type
set val(mac)          Mac/802_11                        ;# MAC type
set val(ifq)          Queue/DropTail/PriQueue     ;# interface queue
type
set val(ll)                 LL                                ;#
link layer type
set val(ant)              Antenna/OmniAntenna         ;# antenna
model
set val(x)                1000             ;# X dimension of topology
set val(y)                1000             ;# Y dimension of topology
set val(cp)               ""              ;# node movement model file
set val(sc)               ""              ;# traffic model file
set val(ifqlen)           50                                ;# max
packet in ifq
set val(nn)               3                                 ;# number
of mobilenodes
set val(seed)             0.0
set val(stop)             1000.0                ;# simulation time
set val(tr)               hidden-out.tr               ;# trace file
name
set val(rp)               DSDV                              ;#
routing protocol
set AgentTrace        ON
set RouterTrace       ON
set MacTrace          OFF

# Open trace file
$ns_ use-newtrace        ;# Use new trace format
set namfd  [open hidden-out.nam w]
$ns_ namtrace-all-wireless $namfd $val(x) $val(y)
set tracefd     [open $val(tr) w]
```

```
$ns_ trace-all $tracefd

# set up topography object
set topo       [new Topography]
$topo load_flatgrid $val(x) $val(y)

# create channel
set chan [new $val(chan)]

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

#  Create the specified number of mobile nodes [$val(nn)] and "attach"
them
#  to the channel. Three nodes are created : node(0), node(1) and
node(2)
      $ns_ node-config -adhocRouting $val(rp) \
                  -llType $val(ll) \
                  -macType $val(mac) \
                  -ifqType $val(ifq) \
                  -ifqLen $val(ifqlen) \
                  -antType $val(ant) \
                  -propType $val(prop) \
                  -phyType $val(netif) \
                  -channel $chan \
                  -topoInstance $topo \
                  -agentTrace ON \
                  -routerTrace ON \
                  -macTrace OFF \
                  -movementTrace OFF

      for {set i 0} {$i < $val(nn) } {incr i} {
            set node_($i) [$ns_ node]
            $node_($i) random-motion 0       ;# disable random motion
      }

# Provide initial (X,Y, for now Z=0) co-ordinates for mobilenodes
#

$node_(0) set X_ 250.0
$node_(0) set Y_ 500.0
$node_(0) set Z_ 0.0

$node_(1) set X_ 500.0
$node_(1) set Y_ 500.0
$node_(1) set Z_ 0.0

$node_(2) set X_ 750.0
$node_(2) set Y_ 500.0
$node_(2) set Z_ 0.0

# Load the god object with shortest hop information
```

```
#
$god_ set-dist 1 2 1
$god_ set-dist 0 2 2
$god_ set-dist 0 1 1

# Now produce some simple node movements
# Node_(1) starts to move upward and then downward
set god_ [God instance]

# Setup traffic flow between nodes   0 connecting to 1 at time 100.0
set udp_(0) [new Agent/UDP]
$udp_(0) set fid_ 1
$ns_ attach-agent $node_(0) $udp_(0)

set udp_(1) [new Agent/UDP]
$udp_(1) set fid_ 2
$ns_ attach-agent $node_(2) $udp_(1)

set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(1) $null_(0)

set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 500
$cbr_(0) set set interval_ 0.005

$cbr_(0) set maxpkts_ 10000
$cbr_(0) attach-agent $udp_(0)

set cbr_(1) [new Application/Traffic/CBR]
$cbr_(1) set packetSize_ 500
$cbr_(1) set set interval_ 0.005

$cbr_(1) set maxpkts_ 10000
$cbr_(1) attach-agent $udp_(1)

$ns_ connect $udp_(0) $null_(0)
$ns_ at 100.0 "$cbr_(0) start"

$ns_ connect $udp_(1) $null_(0)
$ns_ at 100.0 "$cbr_(1) start"



#Define node initial position in nam, only fro nam
for {set i 0} {$i < $val(nn)} {incr i} {
     # The function must be called after mobility model is defined.
     $ns_ initial_node_pos $node_($i) 60
}
# Tell nodes when the simulation ends
for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ at $val(stop)  "$node_($i) reset";
}
```

```
$ns_ at $val(stop)  "stop"
$ns_ at $val(stop)  "puts \"NS EXITING...\" ; $ns_ halt"
proc stop {} {
    global ns_ tracefd namfd
    $ns_ flush-trace
    close $tracefd
    close $namfd
    exec nam hidden-out.nam &
     exit 0
}
puts $tracefd "M 0.0 nn $val(nn) x  $val(x) y $val(y)  rp $val(rp)"
puts $tracefd "M 0.0 sc $val(sc) cp  $val(cp) seed $val(seed)"
puts $tracefd "M 0.0 prop $val(prop) ant $val(ant)"
puts "Starting Simulation..."
$ns_ run
```

OUTPUT:

EX. 4 TOPOLOGY AND MOVEMENT GENERATION

AIM:

To write a tcl script to generate movement of nodes of a given wireless topology.

EXPLANATION:

A simple topology of two nodes is created. UDP CBR traffic is created between the nodes n1 and n0. After initial topology move the two nodes towards each other, when the nodes is within the range data transfer is done, when they move away from each other the data packets are dropped.

PROGRAM:

#Static (Fixed) Topology

```
$node_(0) set X_ 50.0
$node_(0) set Y_ 50.0
$node_(0) set Z_ 0.0
$node_(1) set X_ 150.0
$node_(1) set Y_ 50.0
$node_(1) set Z_ 0.0
$node_(2) set X_ 300.0
$node_(2) set Y_ 50.0
$node_(2) set Z_ 0.0
```

#Node Movement

```
$ns_ at 3.0 "$node_(2) setdest 450 100 50"
$ns_ at 3.0 "$node_(1) setdest 250 100 50"
```

#Grid Topology

```
set val(rlen) 3
set val(nn) [expr $val(rlen) * $val(rlen)]
set gridspace [expr $val(x) / $val(rlen)]
for {set i 0} {$i < $val(rlen) } {incr i} {
for {set j 0} {$j < $val(rlen) } {incr j} {
set a [expr $j + [expr $i * $val(rlen)]]
$node_($a) set X_ [expr 0.0 + [ expr $i * $gridspace]]
$node_($a) set Y_ [expr 0.0 + [ expr $j * $gridspace]]
$node_($i) set Z_ 0.0
}
}
```

#Random Topology

#To apply random movement we need **setdest** command

#Syntax

setdest –n Nunmber of node –p pause time -M Max_Speed -t total_simulation_time –x x-axisvalue

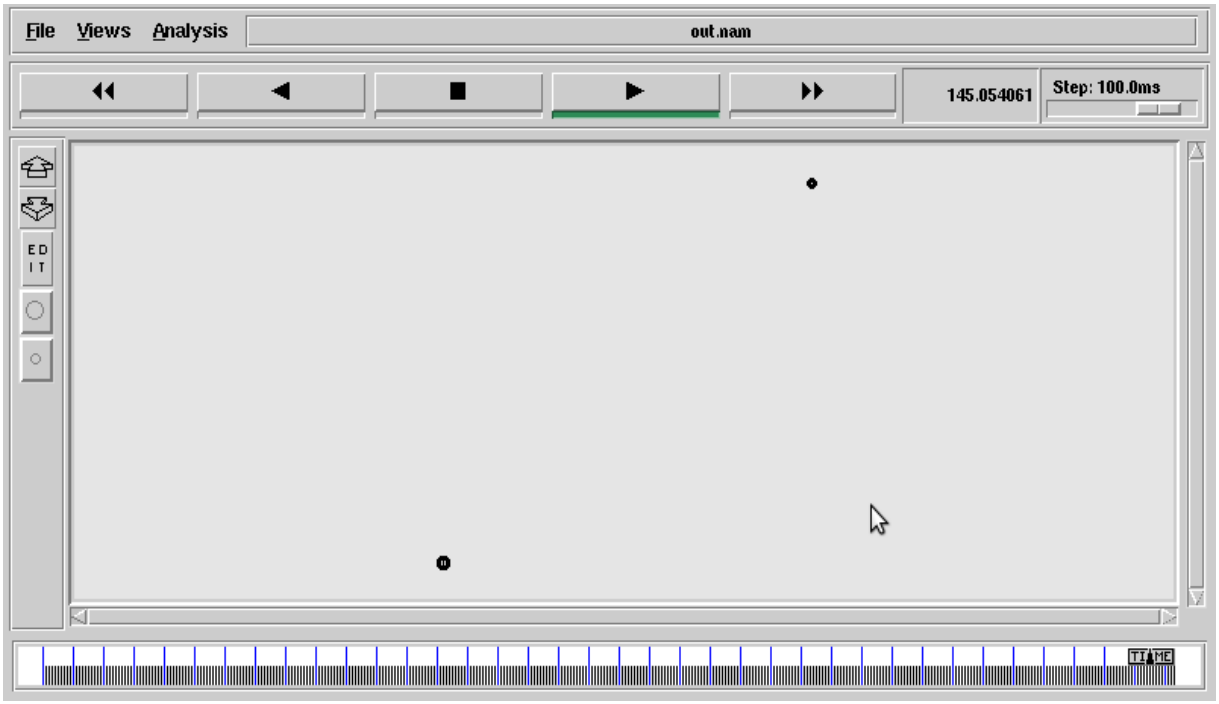
–y y-axis-value

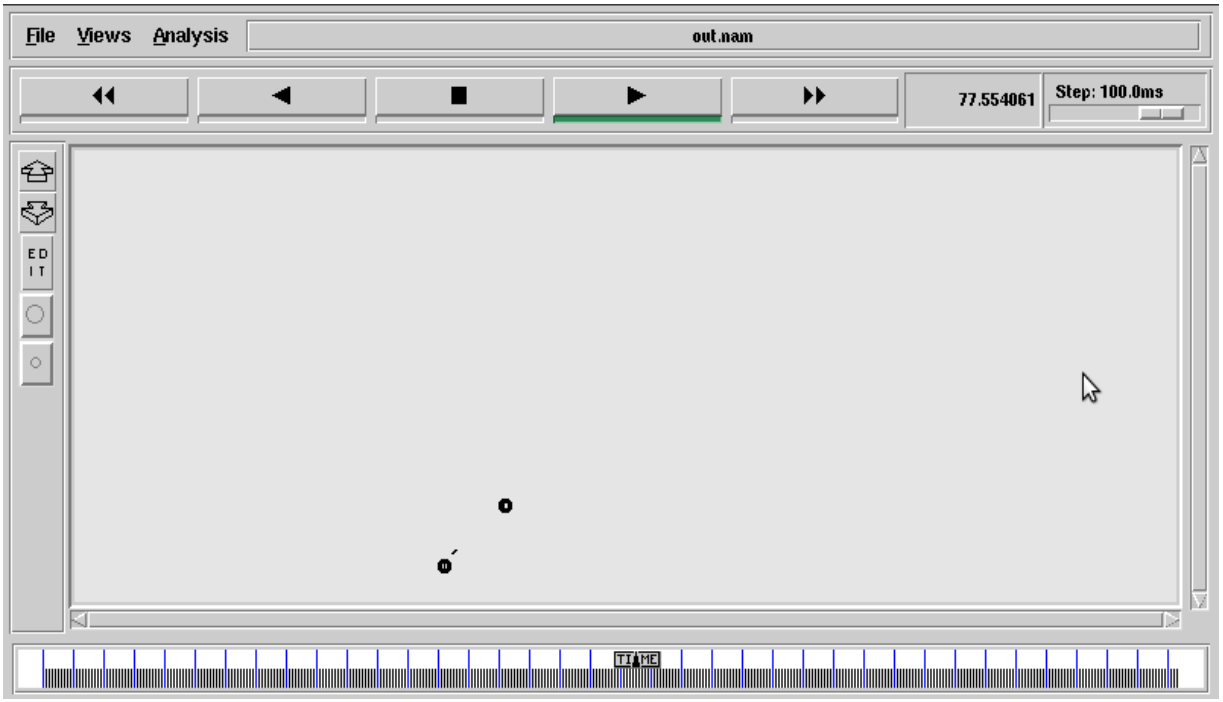
#Example

```
setdest -n 20 -p 10 -M 20 -t 100 -x 500 -y 500 >scen-20-20
setdest -n 30 -p 10 -M 20 -t 100 -x 500 -y 500 >scen-30-20
```

```
setdest -n 40 -p 10 -M 20 -t 100 -x 500 -y 500 >scen-40-20
```

OUTPUT:





EX.5 ENERGY MODEL

AIM:

To create a energy model for a network topology using NS2.

PROGRAM:

```
$ns_ node-config -adhocRouting $val(routing) \
-llType LL \
-macType Mac/802_11 \
-ifqType Queue/DropTail/PriQueue \
-ifqLen 100 \
-antType Antenna/OmniAntenna \
-propType Propagation/TwoRayGround \
-phyType Phy/WirelessPhy \
-channelType Channel/WirelessChannel \
-energyModel EnergyModel \
-rxPower 0.3 \
-txPower 0.6 \
-initialEnergy 10 \
```

EX. 6 WIRED CUM WIRELESS

AIM:
To create a wired cum wireless scenario using ns2 simulator.

EXPLANATION:
A wired cum wireless scenario consists of both wired and wireless topology interlinked and data can be sent from a wired node to a wireless node and vice-versa.

PROGRAM:

```
set val(x) 800
set val(y) 800
set val(ifqlen) 50
set val(tr) wireless.tr
set val(namtr) wireless.nam
set val(nn) 3
set val(adhocRouting) DSDV
set val(stop) 250
set num_wired_nodes 2
set num_bs_nodes 2
set ns_ [new Simulator]
# set up for hierarchical routing
$ns_ node-config -addressType hierarchical
AddrParams set domain_num_ 3
lappend cluster_num 2 1 1
AddrParams set cluster_num_ $cluster_num
lappend no_of_nodes 1 1 4 1
AddrParams set nodes_num_ $no_of_nodes
set tracefd [open $val(tr) w]
$ns_ trace-all $tracefd
set namtracefd [open $val(namtr) w]
$ns_ namtrace-all-wireless $namtracefd $val(x) $val(y)
set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)
```



```
# god needs to know the number of all wireless interfaces
create-god [expr $val(nn) + $num_bs_nodes]
set W(0) [$ns_ node 0.0.0]
$W(0) label [$W(0) node-addr]
set W(1) [$ns_ node 0.1.0]
$W(1) label [$W(1) node-addr]
$ns_ node-config -adhocRouting $val(adhocRouting) \
-llType LL \
```

```
-macType Mac/802_11 \
-ifqType Queue/DropTail/PriQueue \
-ifqLen 50 \
-antType Antenna/OmniAntenna \
-propType Propagation/TwoRayGround \
-phyType Phy/WirelessPhy \
-channelType Channel/WirelessChannel \
-topoInstance $topo \
-wiredRouting ON \
-agentTrace ON \
-routerTrace ON \
-macTrace OFF
```

#create Base Station

```
set BS(0) [$ns_ node 1.0.0]
set BS(1) [$ns_ node 2.0.0]
$BS(0) random-motion 0
$BS(1) random-motion 0
```

#create links between wired and BS nodes

```
$ns_ duplex-link $W(0) $W(1) 5Mb 40ms DropTail
$ns_ duplex-link $W(1) $BS(0) 5Mb 100ms DropTail
$ns_ duplex-link $W(1) $BS(1) 5Mb 50ms DropTail
$ns_ duplex-link-op $W(0) $W(1) orient right
$ns_ duplex-link-op $W(1) $BS(0) orient right-down
$ns_ duplex-link-op $W(1) $BS(1) orient right-up
#configure for mobilenodes
$ns_ node-config -wiredRouting OFF
```

#create Mobile Node

```
set node_(0) [$ns_ node 1.0.1]
```

#connect Mobile Node with Base Station

```
$node_(0) base-station [AddrParams addr2id 1.0.0]
```

#Display Node Address for NAM

```
$node_(0) label [$node_(0) node-addr]
set node_(1) [$ns_ node 1.0.2]
$node_(1) base-station [AddrParams addr2id 1.0.0]
$node_(1) label [$node_(1) node-addr]
set node_(2) [$ns_ node 1.0.3]
$node_(2) base-station [AddrParams addr2id 1.0.0]
$node_(2) label [$node_(2) node-addr]
```



```
$BS(0) set X_ 350.0
$BS(0) set Y_ 600.0
$BS(0) set Z_ 0.0
$BS(0) label [$BS(0) node-addr]
$BS(1) set X_ 350.0
$BS(1) set Y_ 300.0
$BS(1) set Z_ 0.0
```

```
$BS(1) label [$BS(1) node-addr]
$node_(0) set X_ 350
$node_(0) set Y_ 600
$node_(0) set Z_ 0
$node_(1) set X_ 580
$node_(1) set Y_ 600
$node_(1) set Z_ 0
$node_(2) set X_ 620
$node_(2) set Y_ 600
$node_(2) set Z_ 0
for {set i 0} {$i < $val(nn)} {incr i} {
$ns_ initial_node_pos $node_($i) 20
}
$ns_ at 5.0 "$node_(0) setdest 300 300 50"
$ns_ at 5.0 "$node_(1) setdest 300 400 50"
$ns_ at 5.0 "$node_(2) setdest 300 500 50"
```

#create UDP Source

```
set udp0 [new Agent/UDP]
$ns_ attach-agent $W(0) $udp0
```

#create UDP Destination

```
set null0 [new Agent/Null]
$ns_ attach-agent $node_(2) $null0
```

#connecting UDP Source & Destination

```
$ns_ connect $udp0 $null0
```

#create application traffic

```
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
```

#Application start time

```
$ns_ at 1.0 "$cbr0 start"
#Application Stop time
$ns_ at 20.0 "$cbr0 stop"
for {set i } {$i < $val(nn) } {incr i} {
$ns_ at $val(stop).0000010 "$node_($i) reset";
}
$ns_ at $val(stop).1 " $ns_ halt"
$ns_ run
```

EX. 8 ANALYSE THE PERFORMANCE OF A NETWORK USING XGRAPH

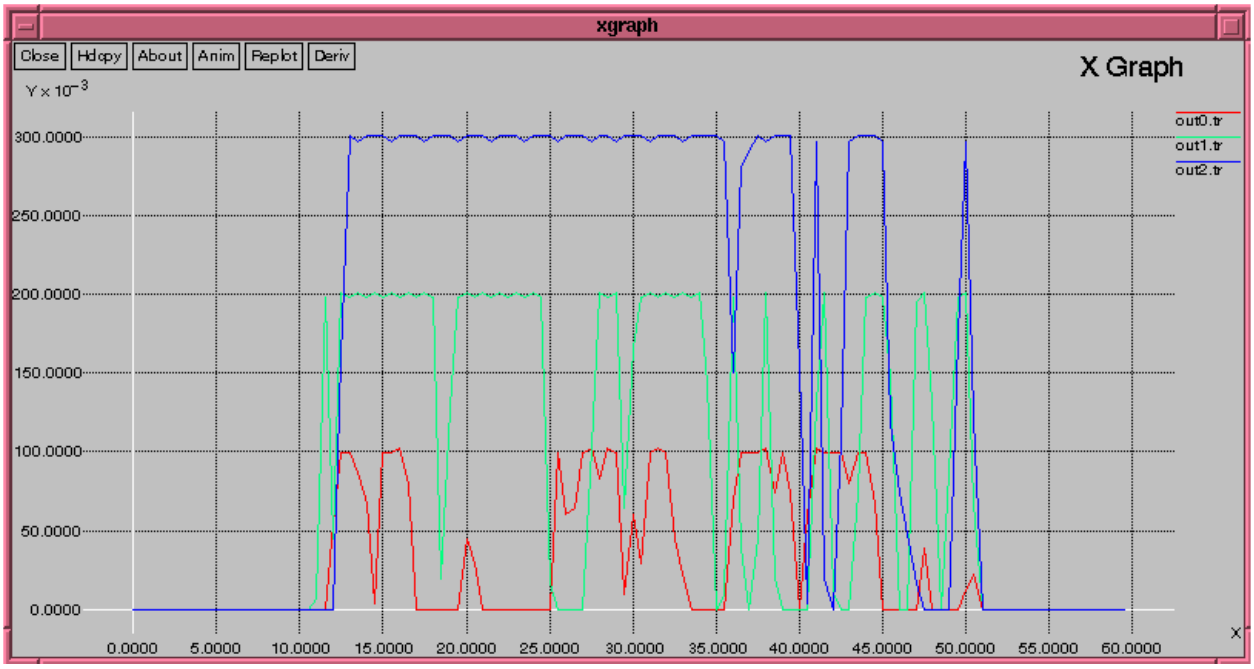
AIM:
To analyse the performance of the given network topology using xgraph.

THEORY:
xgraph is a plotting program which can be used to create graphic representations of simulation results. You can create output files in your Tcl scripts, which can be used as data sets for xgraph. Call xgraph to display the results with the command "**xgraph** <**data-file**>". Below you can see a screenshot of an xgraph window

PROGRAM:

```
#
# Per-packet tracing from node n0 to node n1
#
# create file for monitoring all packets from n0 to n1
set trace_file [open trace_file.out w]
$ns at 0.0 "$ns trace-queue $n0 $n1 $trace_file"
# SimulationTime
set SimTime 10.0
set old_data 0
#Call the procedure "finish" at the end of simulation
$ns at $SimTime "finish"
proc finish {} {
global ns old_data
$ns flush-trace
# Get data from trace file, put current-time and calculated-throughput in a new file
exec awk {
{
if (($1=="-" && $5=="tcp") || ($1=="-" && $5=="cbr")) {
old_data=old_data + $6
print $2, old_data*8.0/$2/1000000
}
}
} trace_file.out > throughput.data
# Call xgraph plotting program to plot throughput vs time
exec xgraph throughput.data &
exit 0
}
```

OUTPUT:


xgraph
Close
Hdcpy
About
Anim
Replot
Deriv
X Graph
Y x 10^-3
300.0000
250.0000
200.0000
150.0000
100.0000
50.0000
0.0000
0.0000
5.0000
10.0000
15.0000
20.0000
25.0000
30.0000
35.0000
40.0000
45.0000
50.0000
55.0000
60.0000
X
out0.tr
out1.tr
out2.tr