R.M.D ENGINEERING COLLEGE

R.S.M. NAGAR, KAVARAIPETTAI – 601206

CW8691

COMPUTER NETWORKS

VI Semester (R-2017)

B.Tech. Computer Science and Business Systems

CW8691 COMPUTER NETWORKS

OBJECTIVES

· To learn and use network commands.

· To learn socket programming.

· To implement and analyze various network protocols.

· To learn and use simulation tools.

· To use simulation tools to analyze the performance of various network protocols.

LIST OF EXPERIMENTS

1. Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute. Capture ping and traceroute PDUs using a network protocol analyzer and examine.
2. Write a HTTP web client program to download a web page using TCP sockets.
3. Applications using TCP sockets like:

· Echo client and echo server

· Chat

· File Transfer

4. Simulation of DNS using UDP sockets.
5. Write a code simulating ARP /RARP protocols.
6. Study of Network simulator (NS) and Simulation of Congestion Control Algorithms using NS.
7. Study of TCP/UDP performance using Simulation tool.
8. Simulation of Distance Vector/ Link State Routing algorithm.
9. Performance evaluation of Routing protocols using Simulation tool.
10. Simulation of error correction code (like CRC).

OUTCOMES

Upon Completion of the course, the students will be able to:

· Implement various protocols using TCP and UDP.

· Compare the performance of different transport layer protocols.

· Use simulation tools to analyze the performance of various network protocols.

· Analyze various routing algorithms.

· Implement error correction codes.

Experiments Beyond Syllabus

1. **Implementation of Stop and Wait protocol and Sliding Window protocol**

EX.NO:1(a) **NETWORKING COMMANDS**

DATE : **(tcpdump, netstat, ifconfig, nslookup and traceroute)**

Aim: To study the different networking commands

- tcpdump
- netstat
- ifconfig
- nslookup and
- Traceroute

Description

0. tcpdump

- tcpdump is a most powerful and widely used command-line packets sniffer or package analyzer tool which is used to capture or filter TCP/IP packets that received or transferred over a network on a specific interface.
- It is available under most of the Linux/Unix based operating systems.
- tcpdump also gives us a option to save captured packets in a file for future analysis.
- It saves the file in a pcapformat, that can be viewed by tcpdump command or a open source GUI

based tool called Wireshark (Network Protocol Analyzier) that reads tcpdump pcap format files.

2. netstat

- The netstat command, meaning network statistics, is a Command Prompt command used to display very detailed information about how our computer is communicating with other computers or network devices.
- Specifically, the netstat command can show details about individual network connections, overall and protocol-specific networking statistics, and much more, all of which could help troubleshoot certain kinds of networking issues.
- Netstat provides statistics for the following:

o **Proto** - The name of the protocol (TCP or UDP).

Local Address - The IP address of the local computer and the port number being used. The name of the local computer that corresponds to the IP address and the name of the port is shown unless the -n parameter is specified. If the port is not yet established, the port number is shown as an asterisk (*).

o **Foreign Address** - The IP address and port number of the remote computer to which the socket is connected. The names that corresponds to the IP address and the port are shown unless the -n parameter is specified. If the port is not yet established, the port number is shown as an asterisk (*).

- Indicates the state of a TCP connection. The possible states are as follows:

State	**Description**
CLOSED	Indicates that the server has received an ACK signal from the client and the connection is Closed
CLOSE_WAIT	Indicates that the server has received the first FIN signal from the client and the connection is in the process of being closed
ESTABLISHED	Indicates that the server received the SYN signal from the client and the session is Established
FIN_WAIT_1	Indicates that the connection is still active but not currently being used
FIN_WAIT_2	Indicates that the client just received acknowledgment of the first FIN signal from the Server
LAST_ACK	Indicates that the server is in the process of sending its own FIN signal
LISTENING	Indicates that the server is ready to accept a connection
SYN_RECEIVED	Indicates that the server just received a SYN signal from the client
SYN_SEND	Indicates that this particular connection is open and active

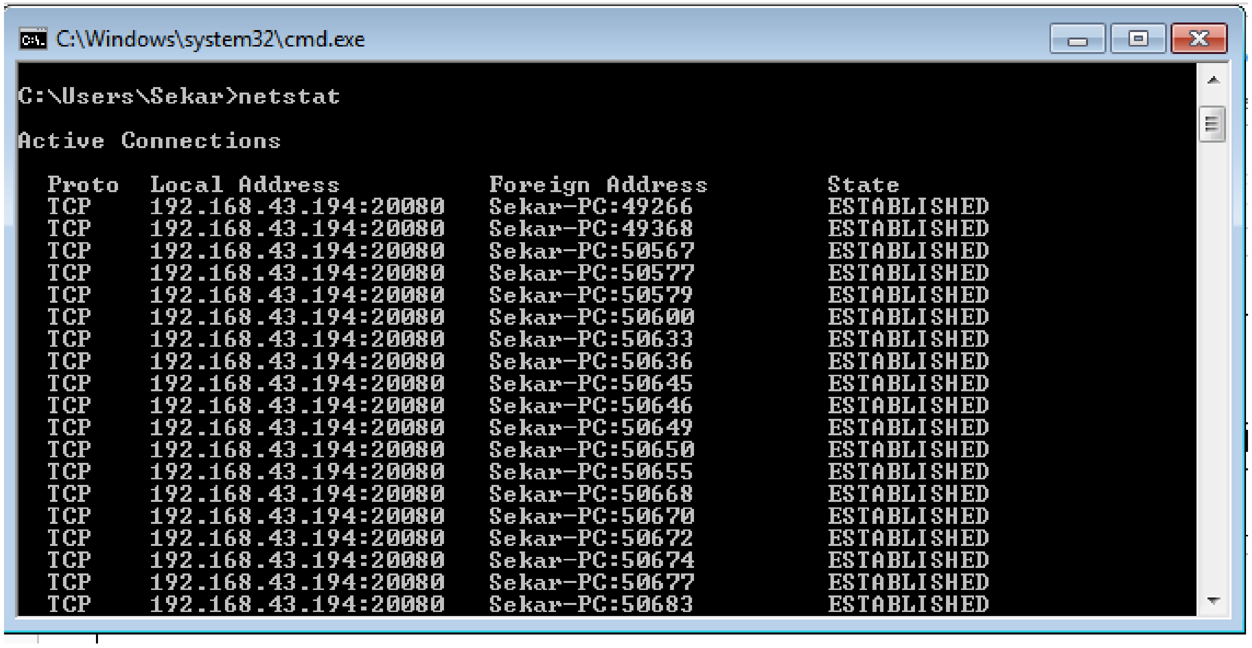
Syntax: netstat [-a] [-e] [-n] [-o] [-p Protocol] [-r] [-s] [Interval]

Used without parameters	**Displays the IP address, subnet mask, and default gateway for all adapters.**
-a	Displays all active TCP connections and the TCP and UDP ports on which the computer is Listening
-e	Displays Ethernet statistics, such as the number of bytes and packets sent and received. This parameter can be combined with -s.
-n	Displays active TCP connections, however, addresses and port numbers are expressed numerically and no attempt is made to determine names.
-o	Displays active TCP connections and includes the process ID (PID) for each connection. You can find the application based on the PID on the Processes tab in Windows Task Manager. This parameter can be combined with -a, -n, and -p.
-p	Shows connections for the protocol specified by Protocol. In this case, the Protocol can be tcp, udp, tcpv6, or udpv6. If this parameter is used with -s to display statistics by protocol, Protocol can be tcp, udp, icmp, ip, tcpv6, udpv6, icmpv6, or ipv6.
-s	Displays statistics by protocol. By default, statistics are shown for the TCP, UDP, ICMP, and IP protocols. If the IPv6 protocol for Windows XP is installed, statistics are shown for the TCP over IPv6, UDP over IPv6, ICMPv6, and IPv6 protocols. The -p parameter can be used to specify a set of protocols.
-r	Displays the contents of the IP routing table. This is equivalent to the route print command.
Interval	Redisplays the selected information every Interval seconds. Press CTRL+C to stop the redisplay. If this parameter is omitted, netstat prints the selected information only once.
/?	- Displays help at the command prompt.

Examples

S. No	**Command**	**Description**
1	netstat -b	To find out which programs are making connections with the outside world
2	netstat -f	Shows all active TCP connections.
3	netstat -o	Display Connection or Ports Process ID
4	netstat -an	The -a parameter lists all the computer's connections and listening ports, while the - n parameter displays addresses and port numbers in numerical format.
5	netstat –s	To find out for any received header error, received address error, discarded packet, etc. It will list out statistics from IPv4, IPv6, ICMPv4, ICMPv6, TCP, UDP, etc.
6	netstat –r	To display Route Table
7	netstat –e	To view the status of all interface. This will display Received & Sent details.
8	netstat –f	Display Fully Qualified Domain Name
9	netstat –n	Displays addresses and port numbers in numerical form.

10	netstat -a	Display All TCP and UDP Connections with Listening Ports



```
C:\Users\Sekar>netstat

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    192.168.43.194:20080   Sekar-PC:49266         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:49368         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50567         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50577         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50579         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50600         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50633         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50636         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50645         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50646         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50649         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50650         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50655         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50668         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50670         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50672         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50674         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50677         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50683         ESTABLISHED
```

3. ipconfig

- Displays all current TCP/IP network configuration values and refreshes Dynamic Host Configuration Protocol (DHCP) and Domain Name System (DNS) settings.
- This command is most useful on computers that are configured to obtain an IP address automatically.
- This enables users to determine which TCP/IP configuration values have been configured by DHCP, Automatic Private IP Addressing (APIPA), or an alternate configuration.
- If the Adapter name contains any spaces, use quotation marks around the adapter name (that is, "Adapter Name").
- For adapter names, ipconfig supports the use of the asterisk (*) wildcard character to specify either adapters with names that begin with a specified string or adapters with names that contain a specified string.
- For example, Local* matches all adapters that start with the string Local and *Con* matches all adapters that contain the string Con.

Syntax

ipconfig [/all] [/renew [Adapter]] [/release [Adapter]] [/flushdns] [/displaydns] [/registerdns] [/shwclassid Adapter] [/setclassid Adapter [ClassID]]

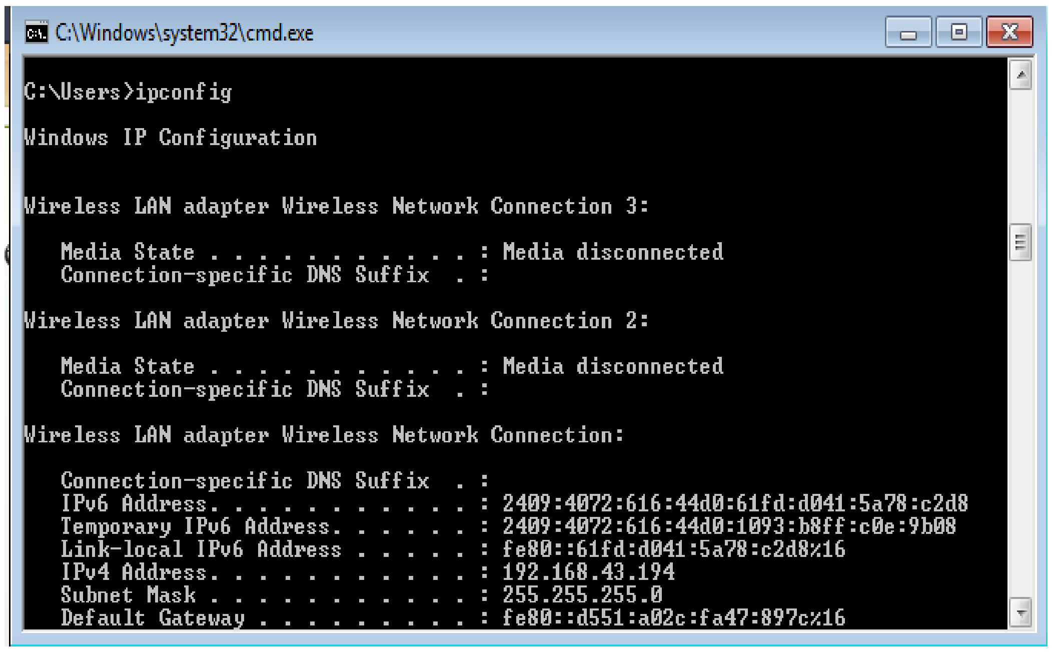
Parameters

Used without

displays the IP address, subnet mask, and default gateway for all adapters.

parameters	
/all	Displays the full TCP/IP configuration for all adapters.
/renew [Adapter]	Renews DHCP configuration for all adapters (if an adapter is not specified) or for a specific adapter if the Adapter parameter is included.
/release [Adapter]	Sends a DHCPRELEASE message to the DHCP server to release the current DHCP configuration and discard the IP address configuration for either all adapters (if an adapter is not specified) or for a specific adapter if the Adapter parameter is included.
/flushdns	Flushes and resets the contents of the DNS client resolver cache.
/displaydns	Displays the contents of the DNS client resolver cache, which includes both entries preloaded from the local Hosts file and any recently obtained resource records for name queries resolved by the computer.
/registerdns	Initiates manual dynamic registration for the DNS names and IP addresses that are configured at a computer.
/showclassid	Adapter Displays the DHCP class ID for a specified adapter. To see the DHCP class ID for all adapters, use the asterisk (*) wildcard character in place of Adapter. This parameter is available only on computers with adapters that are configured to obtain an IP address automatically.
/setclassid	Adapter [ClassID] Configures the DHCP class ID for a specified adapter. To set the DHCP class ID for all adapters, use the asterisk (*) wildcard character in place of Adapter.

OUTPUT



```
C:\Users>ipconfig

Windows IP Configuration


Wireless LAN adapter Wireless Network Connection 3:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Wireless Network Connection 2:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Wireless Network Connection:

   Connection-specific DNS Suffix  . :
   IPv6 Address. . . . . . . . . . . : 2409:4072:616:44d0:61fd:d041:5a78:c2d8
   Temporary IPv6 Address. . . . . . : 2409:4072:616:44d0:1093:b8ff:c0e:9b08
   Link-local IPv6 Address . . . . . : fe80::61fd:d041:5a78:c2d8%16
   IPv4 Address. . . . . . . . . . . : 192.168.43.194
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : fe80::d551:a02c:fa47:897c%16
```

Examples:

ipconfig - To display the basic TCP/IP configuration for all adapters

ipconfig /all - To display the full TCP/IP configuration for all adapters

ipconfig /renew "Local Area - To renew a DHCP-assigned IP address configuration for only the

Connection" Local Area Connection adapter

ipconfig /flushdns - To flush the DNS resolver cache when troubleshooting DNS name resolution problems

ipconfig /showclassid Local - To display the DHCP class ID for all adapters with names that start with

Local

ipconfig /setclassid "Local Area - To set the DHCP class ID for the Local Area Connection adapter to

Connection" TEST TEST

4. nslookup

o nslookup is the name of a program that lets an Internet server administrator or any computer user enter a host name (for example, "whatis.com") and find out the corresponding IP address.

o It will also do reverse name lookup and find the host name for an IP address you specify.

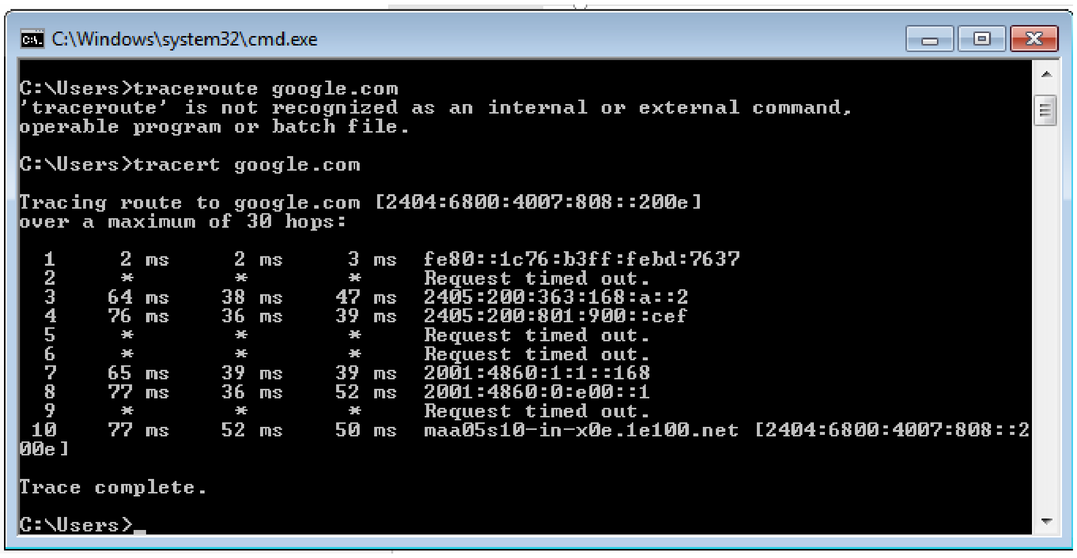
Description	**Command**
Finding The IP Address of an Host	**nslookup** www.google.com
Reverse Lookup IP address to domain name	**nslookup** IP address nslookup 82.165.119.51
Find Mail Servers for a Domain	slookup -querytype=mx domain name nslookup -querytype=mx www.google.com

5. Traceroute

- Traceroute is a network diagnostic tool that displays the route taken by packets across a network and measures any transit delays.

Syntax : tracert domain name
Example : tracert google.com

tracert 64.13.192.208



```
C:\Users>traceroute google.com
'traceroute' is not recognized as an internal or external command,
operable program or batch file.

C:\Users>tracert google.com

Tracing route to google.com [2404:6800:4007:808::200e]
over a maximum of 30 hops:

  1     2 ms     2 ms     3 ms  fe80::1c76:b3ff:febd:7637
  2     *        *        *     Request timed out.
  3    64 ms    38 ms    47 ms  2405:200:363:168:a::2
  4    76 ms    36 ms    39 ms  2405:200:801:900::cef
  5     *        *        *     Request timed out.
  6     *        *        *     Request timed out.
  7    65 ms    39 ms    39 ms  2001:4860:1:1::168
  8    77 ms    36 ms    52 ms  2001:4860:0:e00::1
  9     *        *        *     Request timed out.
 10    77 ms    52 ms    50 ms  maa05s10-in-x0e.1e100.net [2404:6800:4007:808::2
00e]

Trace complete.

C:\Users>_
```

]

Result

Thus a study on different networking commands were made successfully

EX.NO:2 HTTP WEB CLIENT PROGRAM TO DOWNLOAD A WEB PAGE USING TCP SOCKETS

DATE:

AIM:

To write a HTTP web client program to download a web page using TCP sockets in python.

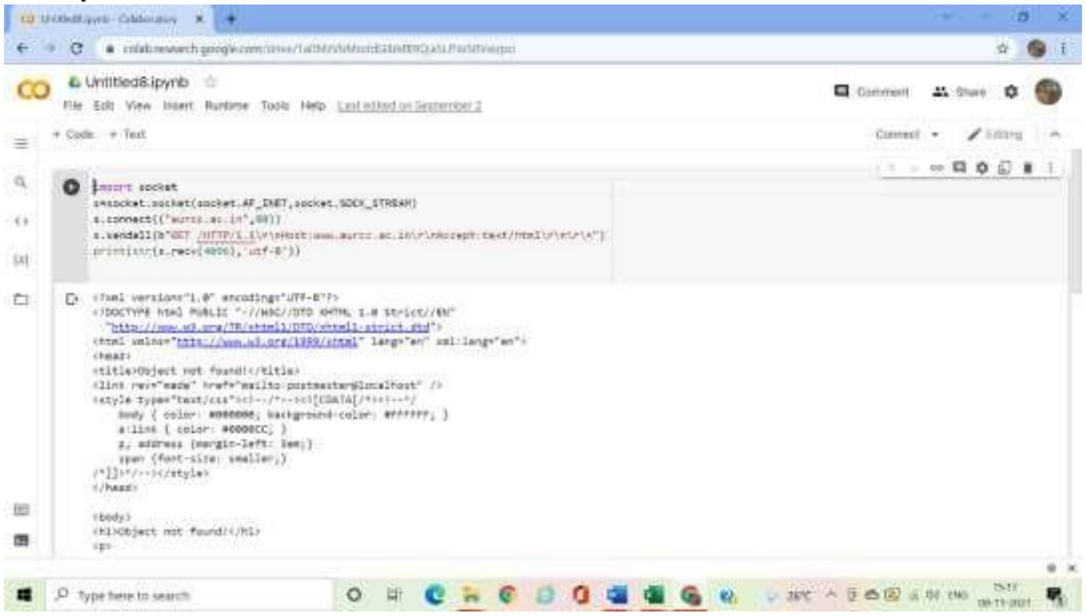
ALGORITHM

- Create socket
- Get server IP address from domain name
- Connect to server using IP address
- Send request to server
- Receive data (webpage)

PROGRAM

```
import socket
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect(("aurcc.ac.in",80))
s.sendall(b"GET /HTTP/1.1\r\nHost:www.aurcc.ac.in\r\nAccept:text/html\r\n\r\n")
print(str(s.recv(4096),'utf-8'))
```

Sample Output



```
import socket
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect(("aurcc.ac.in",80))
s.sendall(b"GET /HTTP/1.1\r\nHost:www.aurcc.ac.in\r\nAccept:text/html\r\n\r\n")
print(str(s.recv(4096),'utf-8'))
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
<head>
<title>Object not found!</title>
<link rev="made" href="mailto:postmaster@localhost" />
<style type="text/css"><!--/*--><![CDATA[/*><!--*/
    body { color: #000000; background-color: #FFFFFF; }
    a:link { color: #0000CC; }
    p, address {margin-left: 3em;}
    span {font-size: smaller;}
/*]]>*/--></style>
</head>

<body>
<h1>Object not found!</h1>
<p>
```

RESULT

Thus the HTTP web client program to download a web page using TCP sockets was executed and output was verified.

EX.NO. 3 (a) TCP SOCKET APPLICATION -ECHO CLIENT AND ECHO SERVER DATE:

AIM

To write a Java program for ECHO using TCP Sockets

ALGORITHM

Client Side Programming

- **Establish a Socket Connection**

∴ 1. To connect to other machine we need a socket connection.

2. A socket connection means the two machines have information about each other's network location (IP Address) and TCP port.

3. The java.net.Socket class represents a Socket.

4. To open a socket: **Socket socket = new Socket("127.0.0.1", 5000)**

5. First argument – IP address of Server. (127.0.0.1 is the IP address of localhost, where code will run on single stand-alone machine).

6. Second argument – TCP Port. (Just a number representing which application to run on a server. ○ For example, HTTP runs on port 80. Port number can be from 0 to 65535)

- **Communication**

1. To communicate over a socket connection, streams are used to both input and output the data.

- **Closing the connection**

1. The socket connection is closed explicitly once the message to server is sent.
2. In the program, Client keeps reading input from user and sends to the server until "Over" is typed.

Client Program:

```
import java.net.*;
import java.io.*;
public class Client
{
// initialize socket and input output streams
private Socket socket = null;
private DataInputStream input = null;
private DataOutputStream out = null;
public Client(String address, int port)
{
        // establish a connection
try
{
socket = new Socket(address, port);
System.out.println("Connected");
// takes input from terminal
input = new DataInputStream(System.in);
// sends output to the socket
out = new DataOutputStream(socket.getOutputStream());
}
catch(UnknownHostException u)
{
System.out.println(u);
}
catch(IOException i)
{
System.out.println(i);
}
// string to read message from input
String line = "";
while (!line.equals("Over"))
{
try
{
line = input.readLine();
out.writeUTF(line);
}
catch(IOException i)
{
System.out.println(i);
}
}
try
{
input.close();
out.close();
socket.close();
}
catch(IOException i)
{
System.out.println(i);
}
}
public static void main(String args[])
{
Client client = new Client("127.0.0.1", 5000);
} }
```

Server Programming

- **Establish a Socket Connection**
  - To write a server application two sockets are needed.
  - A ServerSocket which waits for the client requests (when a client makes a new Socket())
  - A plain old Socket socket to use for communication with the client.
- **Communication**
  - getOutputStream() method is used to send the output through the socket.
- **Close the Connection**
  - After finishing, it is important to close the connection by closing the socket as well as input/output streams.

Server Program:

```
import java.net.*;
import java.io.*;
public class Server
{
//initialize socket and input stream
private Socket         socket = null;
private ServerSocket server = null;
private DataInputStream in        = null;
//  constructor  with  port
public Server(int port)
{
// starts server and waits for a connection try
{
server = new ServerSocket(port);
System.out.println("Server started");

System.out.println("Waiting for a client ...");
socket = server.accept();
System.out.println("Client accepted");
//        takes input from the client socket
in = new DataInputStream(new BufferedInputStream(socket.getInputStream()));
String line = "";
//         reads message from client until "Over" is sent
while (!line.equals("Over"))
{

try
{
line = in.readUTF();
System.out.println(line);
}
catch(IOException i)
{
System.out.println(i);
}                    }
System.out.println("Closing connection");

//        close connection
socket.close();
in.close();
}
catch(IOException i)
{
```

```
System.out.println(i);
}
}
public static void main(String args[])
{
Server server = new Server(5000);
} }
```

OUTPUT

Open two windows one for Server and another for Client

1. First run the Server application as , $ java Server Server started Waiting for a client …
2. Then run the Client application on another terminal as, $ java Client It will show – Connected and the server accepts the client and shows, Client accepted
3. Then start typing messages in the Client window. Here is a sample input to the Client Hello

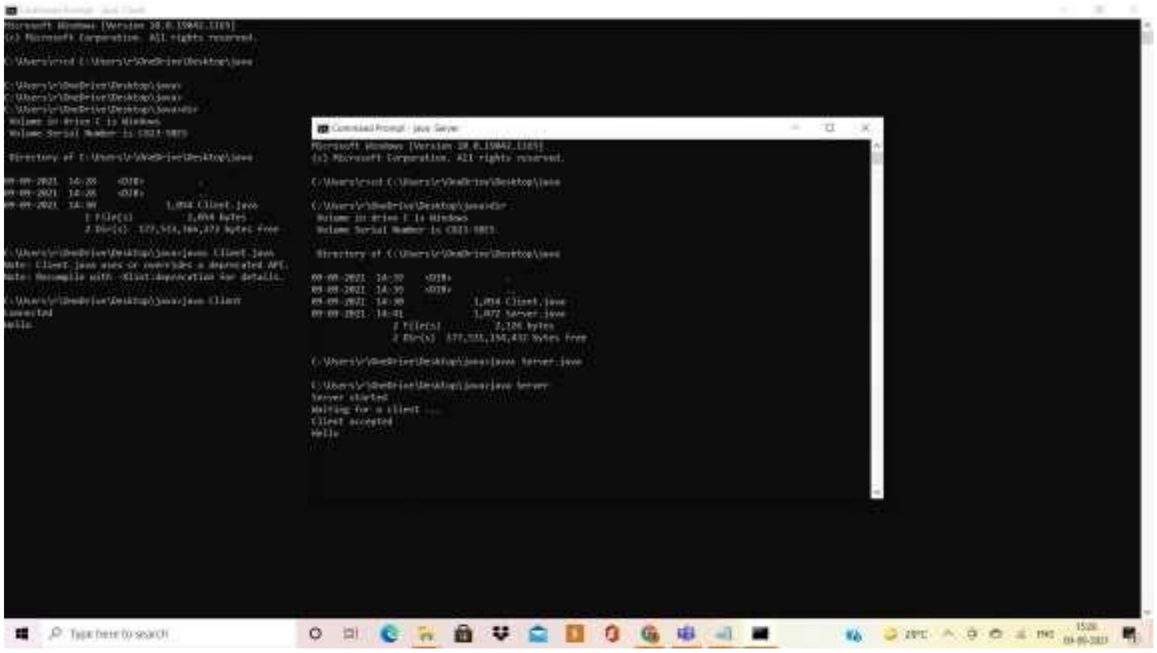
I made my first socket connection Over

Which the Server simultaneously receives and shows, Hello I

made my first socket connection Over

Closing connection

Sending "Over" closes the connection between the Client and the Server just like said before.



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

C:\Users\r>cd C:\Users\r\OneDrive\Desktop\java

C:\Users\r\OneDrive\Desktop\java>
C:\Users\r\OneDrive\Desktop\java>
C:\Users\r\OneDrive\Desktop\java>dir
 Volume in drive C is Windows
 Volume Serial Number is [illegible]

 Directory of C:\Users\r\OneDrive\Desktop\java

09-09-2021  14:28    <DIR>          .
09-09-2021  14:28    <DIR>          ..
09-09-2021  14:30             1,054 Client.java
               1 File(s)          1,054 bytes
               2 Dir(s)  177,514,164,272 bytes free

C:\Users\r\OneDrive\Desktop\java>javac Client.java
Note: Client.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

C:\Users\r\OneDrive\Desktop\java>java Client
Connected
Hello
```

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

C:\Users\r>cd C:\Users\r\OneDrive\Desktop\java

C:\Users\r\OneDrive\Desktop\java>dir
 Volume in drive C is Windows
 Volume Serial Number is [illegible]

 Directory of C:\Users\r\OneDrive\Desktop\java

09-09-2021  14:39    <DIR>          .
09-09-2021  14:39    <DIR>          ..
09-09-2021  14:30             1,054 Client.java
09-09-2021  14:41             1,072 Server.java
               2 File(s)          2,126 bytes
               2 Dir(s)  177,531,156,432 bytes free

C:\Users\r\OneDrive\Desktop\java>javac Server.java

C:\Users\r\OneDrive\Desktop\java>java Server
Server started
Waiting for a client ...
Client accepted
Hello
```

RESULT

Thus the data from client to server is echoed using TCP connection

EX NO : 3(b) **CHAT APPLICATION USING TCP SOCKETS**
DATE :

AIM
To write a java program to initiate a chat between client and server.

ALGORITHM

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

SERVER

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

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

TCPclient1.java

```
import java.net.*;
import java.io.*;
public class tcpclient
{
public static void main(String arg[])
{
Socket c=null;
String line;
DataInputStream is,is1;
PrintStream os;
try
{
c=new Socket( "127.0.0.1",9999);
}
catch(IOException e)
{
System.out.println(e);
}
try
{
os=new PrintStream(c.getOutputStream());
is=new DataInputStream(System.in);
is1=new DataInputStream(c.getInputStream());
do
{
System.out.println("Client:");
line=is.readLine();
os.println(line);
System.out.println("Server:" + is1.readLine());
}
while(line.equalsIgnoreCase("quit")==false);
is1.close();
os.close();
}
catch(IOException e)
{
System.out.println("Socket Closed!Message Passing is over");
}
}
}
```

OUTPUT:

Server Console

```
$ javac tcpchatserver.java
$ java tcpchatserver
Server started
Client connected
Server <<< hi
Message to Client : hello
Server <<< how r u?
Message to Client : fine
Server <<< me too
Message to Client : bye
Client  Disconnected
```

Client Console

```
$ javac tcpchatclient.java
$ java tcpchatclient
Type "end" to Quit
Message to Server : hi
Client <<< hello
Message to Server : how r u?
Client <<< fine
Message to Server : me too
Client <<< bye
     Message to Server : end
```

RESULT

Thus the CHAT program was successfully executed.

Exp. No: 4

Date:

SIMULATION OF DNS USING UDP SOCKETS

AIM

To create a client server program to Domain Name System using the UDP protocol client server.

ALGORITHM

Server

Step1: Start the program.
Step2: Create the socket for the server.
Step3: Bind the socket to the port.
Step4: Listen for the incoming client connection.
Step5: Receive the IP address from the client to be resolved.
Step6: Get the domain name for the client.
Step7: Check the existence of the domain in the server.
Step8: If domain matches then send the corresponding address to the client.
Step9: Stop the program execution

Client

Step1: Start the Program.
Step2: Create the socket for the client.
Step3: Connect the socket to the Server.
Step4: Send the host name to the server to be resolved.
Step5: If the server corresponds then print the address and terminate the process

PROGRAM

UDPclient

```
import java .io.*;
import java.net.*;
classUDPclient
{
public static DatagramSocket ds;
public static intclientport=789,serverport=790;
public static void main(String args[])throws Exception
{
byte buffer[]=new byte[1024];
ds=new DatagramSocket(serverport);
BufferedReader dis=new BufferedReader(new InputStreamReader(System.in));

System.out.println(“server waiting”);
InetAddressia=InetAddress.getLocalHost();
while(true)
{
System.out.println(“Client:”);

String str=dis.readLine();
```

```
if(str.equals(“end”))
break;
```

```
  buffer=str.getBytes();
  ds.send(new DatagramPacket(buffer,str.length(),ia,clientport));

DatagramPacket p=new DatagramPacket(buffer,buffer.length);
ds.receive(p);


   String psx=new String(p.getData(),0,p.getLength());
   System.out.println("Server:" + psx);
   }
   }
   }
```

UDP server

```
import java.io.*;
import java.net.*;
classUDPserver
{
public static DatagramSocket ds;
public static byte buffer[]=new byte[1024];
public static intclientport=789,serverport=790;
public static void main(String args[])throws Exception
{
ds=new DatagramSocket(clientport);
System.out.println("press ctrl+c to quit the program");
BufferedReader dis=new BufferedReader(new InputStreamReader(System.in));
InetAddressia=InetAddress.getLocalHost();
while(true)
{
DatagramPacket p=new DatagramPacket(buffer,buffer.length);
ds.receive(p);
String psx=new String(p.getData(),0,p.getLength());
System.out.println("Client:" + psx);
InetAddressib=InetAddress.getByName(psx);
System.out.println("Server output:"+ib);
String str=dis.readLine();
if(str.equals("end"))
break;
buffer=str.getBytes();
ds.send(new DatagramPacket(buffer,str.length(),ia,serverport));
}
}
}
```

OUTPUT

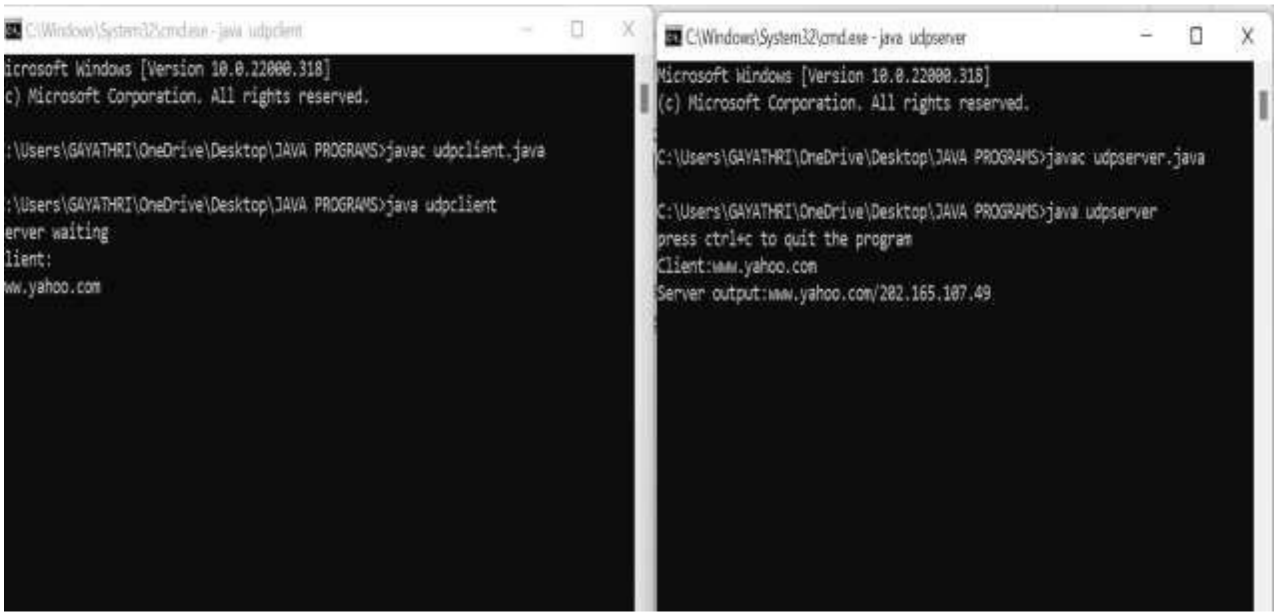
UDPclient

```
D:\Program Files\Java\jdk1.6.0\bin>javac UDPclient.java
D:\Program Files\Java\jdk1.6.0\bin>java UDPclient
Server waiting
Client:www.yahoo.com
```

UDPserver

```
D:\Program Files\Java\jdk1.6.0\bin>javac UDPserver.java
D:\Program Files\Java\jdk1.6.0\bin>java UDPserver
Press ctrl+c to quit the program
Client:www.yahoo.com
Server output:www.yahoo.com/106.10.170.115
```

Output:


C:\Windows\System32\cmd.exe - java udpclient
C:\Windows\System32\cmd.exe - java udpserver

```
icrosoft Windows [Version 10.0.22000.318]
c) Microsoft Corporation. All rights reserved.

:\Users\GAYATHRI\OneDrive\Desktop\JAVA PROGRAMS>javac udpclient.java

:\Users\GAYATHRI\OneDrive\Desktop\JAVA PROGRAMS>java udpclient
erver waiting
lient:
ww.yahoo.com
```

```
Microsoft Windows [Version 10.0.22000.318]
(c) Microsoft Corporation. All rights reserved.

C:\Users\GAYATHRI\OneDrive\Desktop\JAVA PROGRAMS>javac udpserver.java

C:\Users\GAYATHRI\OneDrive\Desktop\JAVA PROGRAMS>java udpserver
press ctrl+c to quit the program
Client:www.yahoo.com
Server output:www.yahoo.com/202.165.107.49
```

RESULT

Thus client server program to Domain Name System using the UDP protocol client server has been executed and verified successfully.

Exp. No: 5A	**SIMULATING ARP PROTOCOL USING JAVA**
Date:	

AIM

To stimulate ARP protocol using JAVA language.

ALGORITHM

Step 1: Import the net, input output, language, utility package s
Step 2: Use the method exec with the command "arp -a " as parameter.
Step 3: Store the result of the ping command in input stream.
Step 4: Process the input stream and display the result.
Step 5: End the process

PROGRAM

```
    import java.net.*;
    import java.io.*;
    import java.lang.Object; // O Should be a capital
    import java.util.*;
    class arp
     {
    public static void main(String args[])
 {
 try
 {
     Process p=Runtime.getRuntime().exec("arp -a");
     Buffered Reader br=new BufferedReader(new InputStreamReader(p.getInputStream()));
     String str,str1="",st1,st2;
       while((str=br.readLine())!=null)
       str1+=str+"\n";
       StringTokenizer st=new StringTokenizer(str1,"\n");
      BufferedReader br1=new BufferedReader(new InputStreamReader(System.in));
    System.out.println("Enter the IP ADDRESS");
     st2=br1.readLine();
    while(st.hasMoreTokens())
       {
     st1=st.nextToken();
    if(st1.indexOf(st2)!=-1)
    System.out.println(st1);
           }
        }
    catch(Exception E)
         {
    E.printStackTrace();
         }}}
```

OUTPUT

```
Enter the IP ADDRESS: 172.15.150.110
00-01-0C-33-1C-5D
```

RESULT

Thus the ARP protocol is successfully completed

.

Exp. No: 5B	SIMULATING RARP PROTOCOL USING JAVA
Date:	

AIM

To stimulate RARP protocol using JAVA language.

ALGORITHM

Step 1: Import the net, input output, language, utility packages.
Step 2: Use the method exec with the command “arp -a “as parameter.
Step 3: Store the result of the ping command in input stream.
Step 4: Process the input stream and display the result.
Step 5: End the process.

PROGRAM

```
import java.net.*;
import java.io.*;
import java.lang.Object; // O Should be a capital
import java.util.*;
class rarp
 {
public static void main(String args[])
   {
try
     {
       Process p=Runtime.getRuntime().exec("arp -a");
       BufferedReader br=new BufferedReader(new InputStreamReader(p.getInputStream()));
       String str,str1="",st1,st2;
       while((str=br.readLine())!=null)
            str1+=str+"\n";
       StringTokenizer st=new StringTokenizer(str1,"\n");
  BufferedReader          br1=new          BufferedReader(new          InputStreamReader(System.in));
  System.out.println("Enter the physical 48-bit ADDR ENTER THE PHYSICAL 48BIT ADDRESS");
      st2=br1.readLine();
while(st.hasMoreTokens())
           {
      st1=st.nextToken();
if(st1.indexOf(st2)!=-1)
System.out.println(st1);
           }
         }
      catch(Exception E)
         {
      E.printStackTrace();
         }   }        }
```

OUTPUT

```
ENTER THE PHYSICAL 48BIT ADDRESS: 00-01-0C-33-1C-5D
172.15.150.110
```

RESULT

Thus the RARP protocols are successfully completed.

Exp. No: 6	STUDY OF NETWORK SIMULATOR (NS)
Date:	

AIM

To study about NS2 simulator in detail.

THEORY

Network Simulator (Version 2), widely known as NS2, is simply an event driven simulation tool that has proved useful in studying the dynamic nature of communication networks. Simulation of wired as well as wireless network functions and protocols (e.g., routing algorithms, TCP, UDP) can be done using NS2. In general, NS2 provides users with a way of specifying such network protocols and simulating their corresponding behaviors. Due to its flexibility and modular nature, NS2 has gained constant popularity in the networking research community since its birth in 1989. Ever since, several revolutions and revisions have marked the growing maturity of the tool, thanks to substantial contributions from the players in the field. Among these are the University of California and Cornell University who developed the REAL network simulator,1 the foundation which NS is based on. Since 1995 the Defense Advanced Research Projects Agency (DARPA) supported development of NS through the Virtual Inter Network Testbed (VINT) project . Currently the National Science Foundation (NSF) has joined the ride in development. Last but not the least, the group of Researchers and developers in the community are constantly working to keep NS2 strong and versatile.

BASIC ARCHITECTURE

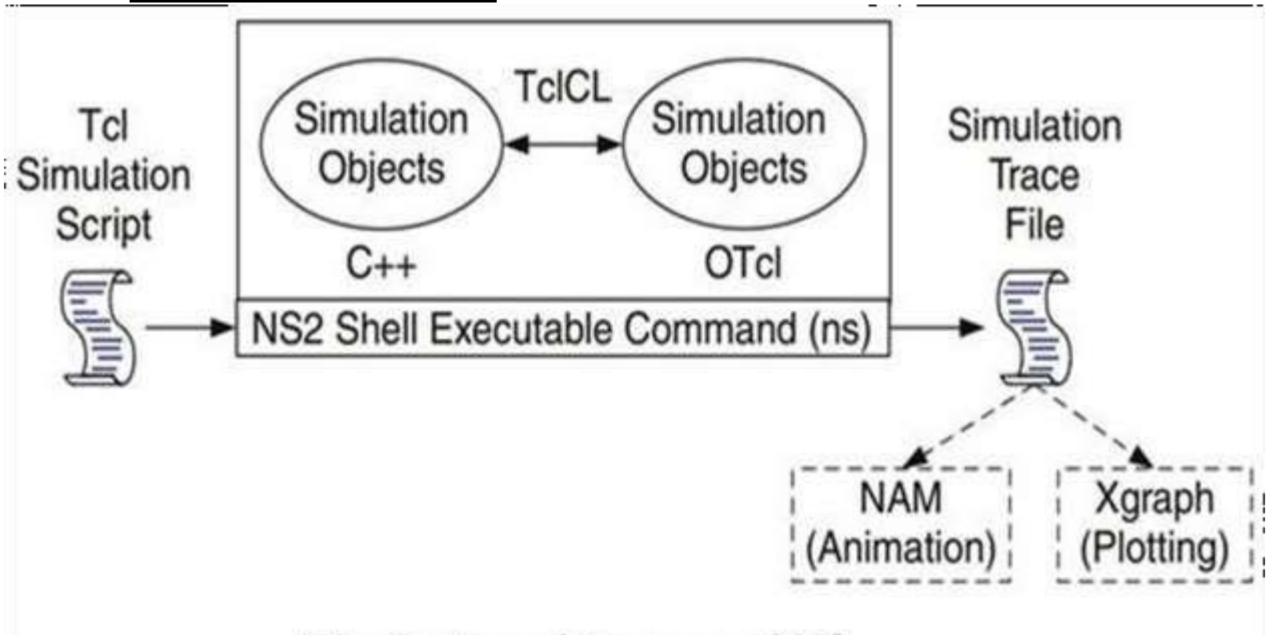


Fig. Basic architecture of NS.

Figure shows the basic architecture of NS2. NS2 provides users with an executable command ns which takes on input argument, the name of a Tcl simulation scripting file. Users are feeding the name of a Tcl simulation script (which sets up a simulation) as an input argument of an NS2 executable command ns.

In most cases, a simulation trace file is created, and is used to plot graph and/or to create animation. NS2 consists of two key languages: C++ and Object-oriented Tool Command Language (OTcl). While the C++ defines the internal mechanism (i.e., a backend) of the simulation objects, the OTcl sets up simulation by assembling and configuring the objects as well as scheduling discrete events (i.e., a frontend).
The C++ and the OTcl are linked together using TclCL. Mapped to a C++ object, variables in the OTcl domains are sometimes referred to as handles. Conceptually, a handle (e.g., n as a Node handle) is just a string (e.g.,_o10) in the OTcl domain, and does not contain any functionality. Instead, the functionality (e.g., receiving a packet) is defined in the mapped C++ object (e.g., of class Connector). In the OTcl domain, a handle acts as a frontend which interacts with users and other OTcl objects. It may define its own procedures and variables to facilitate the interaction. Note that the member procedures and variables in the OTcl domain are called instance procedures (instprocs) and instance variables (instvars), respectively. Before proceeding further, the readers areencouraged to learn C++ and OTcl languages. We refer the readers to [14] for the detail of C++, while a brief tutorial of Tcl and OTcl tutorial are given in Appendices A.1 and A.2respectively.
NS2 provides a large number of built-in C++ objects. It is advisable to use these C++ objects to set up a simulation using a Tcl simulation script. However, advance users may find these objects insufficient. They need to develop their own C++ objects, and use a OTcl configuration interface to put together these objects. After simulation, NS2 outputs either text-based or animation-based simulation results. To interpret these results graphically and interactively, tools such as NAM (Network AniMator) and XGraph are used. To analyze a particular behavior of the network, users can extract a relevant subset of text-based data and transform it to a more conceivable presentation.

CONCEPT OVERVIEW

NS uses two languages because simulator has two different kinds of things it needs to do. On one hand, detailed simulations of protocols require a systems programming language which can efficiently manipulate bytes, packet headers, and implement algorithms that run over large data sets. For these tasks run-time speed is important and turn-around time (run simulation, find bug, fix bug, recompile, re-run) is less important. On the other hand, a large part of network research involves slightly varying parameters or configurations, or quickly exploring a number of scenarios.
In these cases, iteration time (change the model and re-run) is more important. Since configuration runs once (at the beginning of the simulation), run-time of this part of the task is less important. ns meets both of these needs with two languages, C++ and OTcl.

Tcl scripting

Tcl is a general purpose scripting language. [Interpreter]

- Tcl runs on most of the platforms such as Unix, Windows, and Mac.
- The strength of Tcl is its simplicity.
- It is not necessary to declare a data type for variable prior to the usage.

Basics of TCL

Syntax: command arg1 arg2 arg3

Hello World!

puts stdout{Hello, World!} Hello, World!

Variables Command Substitution

```
set a 5 set len [string length foobar]
set b $a set len [expr [string length foobar] + 9]
```

Wired TCL Script Components

Create the event scheduler
Open new files & turn on the tracing
Create the nodes
Setup the links
Configure the traffic type (e.g., TCP, UDP, etc)
Set the time of traffic generation (e.g., CBR, FTP)
Terminate the simulation

NS Simulator Preliminaries.

1. Initialization and termination aspects of the ns simulator.
2. Definition of network nodes, links, queues and topology.
3. Definition of agents and of applications.
4. The nam visualization tool.
5. Tracing and random variables.

Initialization and Termination of TCL Script in NS-2

An ns simulation starts with the command

set ns [new Simulator]

Which is thus the first line in the tcl script. This line declares a new variable as using the set command, you can call this variable as you wish, In general people declares it as ns because it is an instance of the Simulator class,so an object the code[new Simulator] is indeed the installation of the class Simulator using the reserved word new.In order to have output files with data on the simulation (trace files) or files used for visualization (nam files), we need to create the files using ―open command:

#Open the Trace file

```
set tracefile1 [open out.tr w]
$ns trace-all $tracefile1
```

#Open the NAM trace file

```
set namfile [open out.nam w]
$ns namtrace-all $namfile
```

The above creates a dta trace file called out.tr and a nam visualization trace file called out.nam. Within the tcl script, these files are not called explicitly by their names, but instead by pointers that are declared above and called ―tracefile1 and ―namfile respectively. Remark that they begins with a # symbol. The second line open the file ―out.tr to be used for writing, declared with the letter ―w. The third line uses a simulator method called trace-all that have as parameter the name of the file where the traces will go.

Define a "finish" procedure

```
Proc finish { } {
global ns tracefile1 namfile
$ns flush-trace
Close $tracefile1
Close $namfile
Exec nam out.nam &
Exit 0
}
```

Definition of a network of links and nodes

The way to define a node is

set n0 [$ns node]

Once we define several nodes, we can define the links that connect them. An example of a definition of a link is:

$ns duplex-link $n0 $n2 10Mb 10ms DropTail

Which means that $n0 and $n2 are connected using a bi-directional link that has 10ms of propagation delay and

a capacity of 10Mb per sec for each direction.
To define a directional link instead of a bi-directional one, we should replace ―duplex-link by ―simplex-link.
In ns, an output queue of a node is implemented as a part of each link whose input is that node. We should also define the buffer capacity of the queue related to each link. An example would be:

#set Queue Size of link (n0-n2) to 20
$ns queue-limit $n0 $n2 20

FTP over TCP

TCP is a dynamic reliable congestion control protocol. It uses Acknowledgements created by the destination to know whether packets are well received.
There are number variants of the TCP protocol, such as Tahoe, Reno, NewReno, Vegas. The type of agent appears in the first line:

set tcp [new Agent/TCP]

The command **$ns attach-agent $n0 $tcp** defines the source node of the tcp connection.
The command **set sink [new Agent /TCPSink]** Defines the behavior of the destination node of TCP and assigns to it a pointer called sink.

#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]

$ns attach-agent $n5 $null
$ns connect $udp $null
$udp set fid_2

#setup a CBR over UDP connection
The below shows the definition of a CBR application using a UDP agent
The command **$ns attach-agent $n4 $sink** defines the destination node.
The command **$ns connect $tcp $sink**
finally makes the TCP connection between the source and destination nodes.
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set packetsize_ 100
$cbr set rate_ 0.01Mb
$cbr set random_ false
TCP has many parameters with initial fixed defaults values that can be changed if mentioned explicitly. For example, the default TCP packet size has a size of 1000bytes.This can be changed to another value, say 552bytes, using the command **$tcp set packetSize_ 552**.
When we have several flows, we may wish to distinguish them so that we can identify them with different colors in the visualization part. This is done by the command **$tcp set fid_ 1** that assigns to the TCP connection a flow identification of ―1.We shall later give the flow identification of ―2‖ to the UDP connection.

RESULT

Thus the Network Simulator 2 is studied in detail.